Department of Computer Science

Technical Report

Undergraduate Dissertation: Developing an Artificially Intelligent Real Time Blues Accompanist

Andrew Brothwell

Technical Report 2007-01 **March 2007**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

DEVELOPING AN ARTIFICIALLY INTELLIGENT REAL TIME BLUES ACCOMPANIST

Andrew Brothwell

BSc (Hons) in Computer Science

University of Bath

2006

DEVELOPING AN ARTIFICIALLY INTELLIGENT REAL TIME BLUES ACCOMPANIST

submitted by Andrew Brothwell

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

……………………….

……………………….

Abstract

This paper presents the development of a computer musical accompanist. This system attempts to model some of the characteristics of human musicianship, particularly concentrating on the human sense of rhythm, and the playing ‘by ear’ skill set. It specifically focuses on blues accompaniment, and works with no previous knowledge of the particular song that is about to be played. The performance of the lead musician is constantly monitored and reacted to throughout the song.

Acknowledgements

I would like to thank Professor John Fitch for the guidance and expertise that he provided me with throughout the course of this project. Thanks also go to the friends and musicians that helped out with the testing of the system.

Contents

1 Introduction

An issue that faces many musicians is that practising and playing their instruments alone can become dull and unfulfilling. This is particularly true with musicians playing ‘lead’ instruments in styles that would usually be played as part of a full band. These can often sound empty when played alone without backing chords or accompaniment.

Currently Musicians have two options if they want to play their instrument along with accompaniment. They can join a group of musicians; human musical accompanists know how to listen to the lead musician, and play in the key and the tempo that the lead musician chooses. In most cases this is the preferred choice, and will most likely be so for many years to come, however there is another option; artificial accompaniment.

Artificial musical accompaniment has been around for many years in various forms, although not necessarily computer based or using any artificial intelligence. Although the versatility and intelligence of an artificial accompaniment will not match its human counterpart, there are many practical advantages. It will be available at any time, will not cost anything significant to run, will be infinitely patient, and will never tire.

At the most basic levels of artificial accompaniment are simply compact discs containing compilations of pre-recorded music, with certain musical parts not included in the mix. This allows players of that instrument to fill in the missing part. This is arguably not accompaniment at all as it is the musician who must listen and follow the backing band. Despite this limitation this type of product is very popular with musicians, as they give at least some impression of playing with others.

In trying to gain versatility from this type of accompaniment, many musicians use products such as the ‘training c.d. player’. This is in effect a c.d. player that can adjust the pitch and tempo of the music independently. This allows a pre recorded backing track to be played in different keys and at different speeds to suit the musician. The fact that products like this are popular shows how much potential interest there is in even the most basic artificial accompanists.

The problem with pre-recorded accompaniment is that it is, in many ways, the antithesis of the ideal musical model; the accompaniment should follow the lead instrument, not the other way around. For many years computer scientists have been looking at the problem of developing musical accompanists with artificial intelligence. By “listening” to a lead instrument, and analysing the pitch and tempo of the melody being played, the accompanist can become reactive.

Most of these existing systems concentrate on matching up a given score with a lead musician; they already know the song and just try and play along at the right time. So is it possible to develop a system that can do this without a score, working solely on audio input and without specific knowledge of the song about to be played?

A real-time system like this, which models a human musician playing ‘by ear’, would be breaking new ground in this field. It will require the analysis of the thought processes that a musician follows when accompanying a lead instrument without written music. The goal will then be to produce a computer program that attempts to model this as closely as possible.

Blues is a type of music famous for being improvised; the accompaniment of blues music without a score is the standard rather than the exception. One of the first things that a blues musician will learn is how to play along with a twelve bar blues song. As the role of accompanying musician is so clearly defined in this type of music, it would seem to make a perfect starting point for modelling human behaviour in a system like this.

1.1 Aim

To develop a computer based, artificially intelligent blues accompanist system. It should work in real-time and produce backing music that follows a lead musician playing a twelve-bar blues song. It should where possible react to the lead musician in the same ways as a human accompanist might.

2 Literature Survey

2.1 Introduction

The purpose of this review is to research the current methods and technology that have already been developed relating to computer musical accompaniment. It is important to gain a broad understanding, as well as to specifically research existing projects of a similar nature, so that lessons may be learnt from others and adapted for use in this project. The research areas relating to this project are broadly split into two areas; the environment interaction, and the artificial intelligence.

This system aims to model a human musical accompanist, and as such it must be able to interact with its environment in the same way as a human would. This means that it must be able to 'hear' the pitch and tempo of the melody played by the musician. It must also have the ability to produce its own sound in the form of backing music to accompany the melody. Current technologies in pitch and tempo detection, as well as in instrument synthesis must be researched.

The artificial intelligence that will calculate the backing music to play along with the lead musician will be the most interesting and challenging part of the system to develop. Once we have the ability to 'hear' the melody, we need to program an artificially intelligent accompanist that reacts to this in as similar way as possible to a human musician. We therefore must research and fully understand the techniques that will be required to do this. An appropriate development environment for this system will also be chosen, and the reasoning behind the choice given.

2.2 Existing Computer Musical Accompanists

The first example of a successful reactive real time musical accompanist was demonstrated at the 1984 International Computer Music Conference, where Roger Dannenberg demonstrated his system (Dannenberg, 1984). He produced an algorithm that matched up a musicians live performance with a score that had been previously programmed in. The results were fairly successful, and he had at least partially mimicked a human musician.

In 1989 Dexter Morrill, a university professor of music, developed an accompanist that could play along with a trumpet, this time without having to have an entire score for the song programmed into it. This system relied on switches under the valves of the trumpet to convert the instrument into a MIDI controller. It also required some information to be entered about time and key before it could play, but was still very impressive. It played along in real-time, and could create accompaniment based on the notes recently used by the musician.

Possibly the most advanced real time accompanist to date was developed recently by Christopher Raphael, and named 'Music Plus One'. This system very accurately played classical musical accompaniment from a score, listening and being led by a lead instrument, and cleverly matching up the lead musician's performance with the correct part of the score.

An existing system that interests me considerably is the 'Subsumption Strategy Development of a Music Modelling System' (Bryson, 1992). This project aimed to achieve chord prediction to accompany a single note melody, and did so with some success. This system shares our interest in attempting to model the behaviour of a human accompanist playing 'by ear'. It tries to derive backing for a melody, with no previous knowledge of the song, and is probably the existing system that lies closest to what we are trying to achieve in this system. Although it never worked in real time, many of the ideas used in the program would translate perfectly well to a real time system. It is therefore worth examining this system in some detail.

In 'Subsumption Strategy' each task is broken down into a layer; there are the low level layers that interpret their environment, and the higher level layers that take the outputs of the low level layers and use them to achieve a high level goal. The problem of a reactive musical accompanist was suited to this technique, as it needed to pursue multiple goals simultaneously, playing both in correct time and pitch. It also needed to be aware of events external to it, and process sensor data, processes particularly suited to low level layers in this architecture.

In adapting the strategy to a musical accompanist Joanna Bryson split the system into the following independent competencies:

- Note – The pitch detector. It output a weighted list of notes that had occurred within various overlapping time frames. An example of a low level module.

- Chord – This module took a weighted list of notes (such as the output from 'note') and outputted a weighted list of the most likely chords that would match those notes. This was therefore a higher level module.

- Thresh - Thresh looked at pitch stability and outputted what it considered to be the times at which notes changed. A low layer module.

- Beat - Beat timed the intervals between these changes, and returned a list of the most common durations between them, therefore estimating the beat length. It was a higher level module.

- Timed – This was a higher level module still, and took the information provided by 'Beat', using this information to predict future beats.

There are ideas in this system that may well be applicable to the one we are attempting to develop. The strategy of breaking down the system into individually solvable sections seems sensible and appealing.

2.3 Blues Accompaniment

The blues is a vocal and instrumental musical form, originally derived from African-American work songs. Blues music has had a major influence, finding expression in jazz, big bands, rhythm and blues, rock and roll and country music as well as conventional pop songs and even classical music.

Early forms of the blues evolved in the southern parts of the United States in the late 19th and early 20th centuries, using simple instruments such as acoustic guitar, piano, and harmonica. Songs were predominantly based on a twelve bar structure using tonic, subdominant and dominant chords. Melodically, blues music is marked by the use of the flattened third and dominant seventh (so called blue notes) of the associated major scale. The blues scale is also frequently used in non-blues musical forms (Davis, 2003).

An important aspect of blues music is that although it can vary greatly within its structural constraints, chord patterns and scales are strongly dictated. This is significant as it allows skilled blues musicians, even if they have never met, to easily play together with no pre-determined musical score or knowledge of the song. This gives great scope for musical improvisation within a song, a major part of playing the blues. It is very unusual for a blues musician to know exactly what they are going to play themselves before they start a song.

Many famous blues musicians, from the old Mississippi Delta blues men such as Son House, right up to the more modern rock and roll style blues men such as Chuck Berry would tour without their own backing band. They would simply hire in local blues musicians in each new town. This demonstrates that blues music can be played to the very highest of standards when the backing band has practically no prior knowledge of the songs. It is this aspect that makes it an ideal musical style to use as a starting point for this accompanist system.

2.4 Development Environment

The overall aim of this project is to implement a musical accompanist, and as such one would aim to spend the majority of the development time working on the artificial intelligence behind this. It would be of great benefit therefore, to work with a language that has the in-built ability to output musical sounds as and when they are needed, and to analyse audio input.

CSound is a programming language specifically designed for the production and manipulation of sound and music. It has two sub-languages, the orchestra language, and the score language. The orchestra is used to specify the instruments playing in a piece of music, whilst the score is used to specify when and with what properties (i.e. pitch, amplitude, duration) each instrument will play (Boulanger, 2000). The language can both output sound directly to the soundcard, and take input from a microphone, which makes it suitable for developing a real-time project such as this one. Within CSound instrument modelling is a widely researched and practiced field. This means that information will be readily available to aid in the design of the backing instrument sounds.

There are lots resources on the Internet for Csound and there is also the added advantage that the project supervisor, Prof. John Fitch is a leading expert in the language. These things make Csound the obvious choice for this project.

2.5 Pitch Detection

Pitch detection will be a vital component of the accompanist system. The project will depend upon the ability to tell which note the musician is playing. This must be done both quickly and accurately. Pitch detection is a well researched area, but it is not ‘solved’ - there is not yet one generally accepted best method. It is important therefore to research the topic adequately, and to find a suitable method to use for this accompanist system.

A real world waveform will usually consist of many ‘partials’, a set of different and often harmonically related frequencies, rather than just a single frequency. There will exist a ‘fundamental frequency’, which is usually the lowest of these partials that relates well to most of the other partials, i.e. is linked by small whole-number ratios.

Pitch is a perceptual quantity related to this ‘fundamental frequency’ of a periodic or pseudo-periodic waveform. The musical scales and recognised pitches used today were developed before people knew about frequency and spectral content, and were based on the similarities and dissimilarities between the notes. This understanding of pitch is based on the log of the frequency, with each octave increase being a doubling. The way that people perceive pitch is in fact more complex than this though; frequency doubling in very low notes corresponds to a pitch interval slightly less than an octave, while doubling in high ranges corresponds to an interval slightly more than an octave. It is perhaps more accurate to say that the component we need is a fundamental frequency detector, but ‘pitch detection’ remains the commonly used term.

The difficulty in finding this fundamental frequency of a waveform depends on the waveform itself- if the waveform has fewer partials, or the power of the partials is small, the fundamental frequency is often easier to detect. If the partials have more power, especially if they have more power than the fundamental frequency itself, then the period can be much harder to detect.

There are a number of standard methods that researchers use to extract the pitch, based on various mathematical principles and techniques. It is notoriously difficult to compare the performance of these pitch extraction algorithms for several reasons. Firstly performance depends on domain, for example one algorithm may perform very well on a guitar signal, but very poorly on a human voice. Secondly, it is difficult to rate the results of the pitch extractor, precisely because it is difficult to measure the fundamental frequency in the first place.

Existing algorithms can be split into two general groups; time domain methods and frequency domain methods (Gerhard, 2003). We will look at some of the different types of pitch extraction algorithms in the sections below.

2.5.1 Time Domain Methods

Time domain methods analyse the waveform's amplitude as it changes with time. These methods tend to be the more basic approaches to pitch extraction, and are also computationally the least expensive.

There are a group of time domain methods that work by attempting to discover how often a waveform fully repeats itself. These methods can be tailored to a particular type of waveform; an event that is known to occur once every waveform can be identified and counted.

One of these is the zero-crossing rate (ZCR) method, which simply counts how often the waveform crosses zero displacement per unit of time. The problem with this method is that if the waveform contains high frequencies it will often cross zero several times per cycle. A reasonable solution is to apply a low pass filter to the waveform before analysing it to get rid of these high frequencies. The cut-off for this filter must be chosen carefully however, and will have to be changed to suit different types of sounds. Another solution is to look for patterns in the zero-crossings to account for several zero-crossings per cycle.

Another similar approach is the peak rate method. This counts the number of peaks in the waveform per unit of time. Theoretically only maximums (or minimums) must be counted, as one will occur each cycle. Similar problems with this method can occur when the waveform has more than one maximum per cycle.

The slope event rate method is another similar technique; it assumes that if the waveform is periodic, then its slope must also be periodic. For some cases looking at the changes in slope can be more informative and easier to interpret than looking at the zeros and peaks in the waveform.

Autocorrelation is a time domain method that takes a slightly different approach to previously discussed algorithms. Rather than just looking for characteristics in the waveform itself, this method looks at the correlation between the waveform and the same waveform after a certain delay or 'lag'. When this lag of a periodic waveform is the length of half a cycle the correlation will reach a minimum, as the delayed wave and the original wave will be out of phase. As the lag increases to the length of an entire waveform, the correlation will reach a maximum, as the two waves will be in phase. This first maximum in autocorrelation signifies the period of the waveform.

A further development of autocorrelation is the 'average magnitude difference function' (AMDF) (Ying, Jamieson and Michell, 1996). This is similar to autocorrelation in that it compares the waveform with a delayed version of itself, but this time it looks at the average of the square root of the differences between the two. As a square root is a non-linear compressor this gives a 'sharpened' result, increasing the contrast where it is most needed. This helps with reducing the effect of noise etc.

Time domain methods tend to be simple to understand and implement, and are computationally inexpensive. If the nature of the waveform they are analysing is understood, they can easily be tailored to suit that waveform, increasing the accuracy.

The major setbacks of many time domain methods result from the fact that the majority of waveforms rarely have just one event per cycle, be it zero-crossings, maximums, or slope events. There are measures to reduce the errors caused by this, but they tend make the pitch detector only suited to specific types of waveform.

2.5.2 Frequency Domain Methods

Where as with time domain methods we were looking as the waveform as the change in air pressure over time, there are a group of pitch detectors that look at the frequency domain, where the time axis is replaced by frequency. This requires Fourier decomposition of the signal. This group of methods tend to be more versatile and produce more accurate results, but are computationally far more expensive.

A frequency domain method pioneered by Martin Piszczalski as part of his automatic music transcription system (Piszczalski, 1986) uses component frequency ratios to find the fundamental frequency of a signal. Once in the frequency domain the method uses peak detection to identify the partials, and then looks at the ratios between pairs of these partials to estimate the fundamental frequency. The interesting thing with this type of method is that the fundamental frequency itself need not be present in the set of partials for it to be calculated correctly.

Filter based methods are commonly used in the frequency domain for pitch detection. A comb filter is one that has many equally spaced pass bands. The ‘optimum comb filter’ method applies a number of comb filters based on differing frequencies. When a signal is made up of evenly spaced harmonics, the comb filter whose spacing most closely matches these harmonics will have a much higher output, and we will know the corresponding fundamental frequency of that filter. A problem with the ‘optimum comb filter’ method is that if there is just one partial in the signal, the fundamental frequency itself, it will fail. This is because any of the comb filters with a pass band at that frequency will produce the same output. Some newer filter based methods use sweeping ‘tuneable’ filters, which are swept across the frequency spectrum until the point at which a maximum output is obtained.

Another frequency domain method is ‘cepstrum analysis’. This looks at the Fourier transform of the log of the magnitude spectrum of the input waveform, the idea being to provide a linear view of a non-linear problem. Once this transform has been applied to the signal, there should be a single maximum in the output that relates to the fundamental frequency.

The harmonic product spectrum (HPS) method compresses the spectrum a number of times by small whole number ratios, and compares with the original spectrum (Noll, 1969). The first peak in the original spectrum will coincide with the second peak in the spectrum compressed by a factor of two, which coincides with the third peak in the spectrum compressed by a factor of three. Hence, when the various spectrums are multiplied together, the result will form clear peak at the fundamental frequency.

Overall frequency domain pitch detection algorithms are very versatile and can perform well even for polyphonic inputs with far greater success that time domain methods. The drawback is that the signal must be converted into the frequency domain in the first place, usually via a fast Fourier transform (FFT). This makes the process far more computationally expensive. These methods also tend to be more complex and therefore more difficult to program.

2.5.3 Conclusion

After looking at many different pitch detection algorithms, the conclusion has been reached that a time-domain method would be more suited to this project. In nearly all previous cases of projects requiring real-time pitch extraction, it has been found that people have avoided converting to the frequency domain. Not only would we risk the pitch detection working too slowly, which would make the accompanist useless, but we would also be setting ourselves a far more complicated programming task.

Within time domain methods it seems that the autocorrelation and AMDF methods have been the most popular. They are simple enough computationally to work in real time, but achieve far greater accuracy than the simple wave event methods (ZCR, peak, slope event). Helpfully there is already an AMDF pitch extractor built in to Csound. This would appear to be by far the best method to use, and the fact that a ready made version already exists makes this the method of choice beyond doubt.

2.6 Tempo Extraction and Beat Tracking

This musical accompanist project will rely heavily on being able to detect tempo accurately. In terms of a human, the first indication of an accompanist that lacks competence is one that cannot stay in time. If we can program a method of tempo detection accurate enough to allow our system to play in correct time with the musician, then the whole system will sound far more realistic and professional.

In fact the problem of tempo detection is twofold. We have to establish the speed that the musician is playing at; this is usually measured by musicians in beats per minute (BPM). On top of this, however, we must calculate the actual locations in time of these beats. In other words, to be able to predict when future beats will occur, we not only need to know the gaps between beats, but also the time at which the beats occurred.

There are several different methods used in tempo detection, and we will need to analyse a range of them before deciding upon which one to use. The method that we choose must work effectively in real time, and give all the information we need to allow us to predict when the next beat will occur.

An onset can be defined as the start of a sound event (Lerch, 2005). Specifically in music this is usually the start of a note being played, or a percussive instrument being struck. Finding the location of the onsets in the input audio can solve the problems of both tempo extraction and beat location tracking.

The problem, then, is to find a suitable method for detecting these onsets. This is a fairly well researched field, but we need to find a method that is suited specifically to this project. The method must work fast enough in real time for me to use the data and predict the time location of the next beat before it occurs. It must also be relatively simple to implement, as there will be time limitations when programming this project. Another important consideration is that we must expect input that contains no percussion, so there may not be very clear amplitude peaks at the start of notes.

Many of the methods used for tempo extraction can be directly related to methods described previously for pitch detection, although they look at far larger time samples; they are looking for periods between peaks in entire notes rather than individual wave peaks (Scheirer, 1997).

A very simple onset detector that could be used would be similar to the peak rate method described for pitch extraction. The values of displacements in the waveform are squared and rooted so that negative values become positive. This then gives larger values for higher amplitudes, and taking an average of these values across a time period gives the ‘root mean squared’ – a measure of the loudness of the audio during that period. The time at which the root mean squared (RMS) of the input audio reaches a level above a certain threshold is recorded as an onset.

As the input will have no percussion, this method may only work well if the instrument has a fast decay rate and hence clear peaks in amplitude when a note is initially played. It might, for instance, work well with an acoustic guitar, but not with a cello, which is not plucked and so would have less volume variation at a note’s onset.

One system that has been proposed for measuring tempo of a drumless audio signal in real time uses entirely pitch change as a measure of onsets (Goto, 1998). This system also shows some interesting algorithms for predicting the next beat time, based on previous beats. It constantly checks the beat times it previously predicted against recorded beat times, and adjusts adapts its future beat predictions to account for any error.

2.6.1 Conclusion

Having evaluated the various methods of onset detection there are a few that would be feasible for this project. There is no onset detection feature built into CSound and so whatever method we do choose will have to be implemented form scratch.

In terms of performance, constantly monitoring the RMS of the input signal will be far less computationally expensive than continually calculating its pitch. With this in mind, the simple RMS analysis approach might be the most appropriate. If the pitch detection method turns out to be fast enough to run constantly, then it might also be worth trying the pitch change method.

2.7 Music Generation

Although the focus of this project will be on the processes of deciding what to play and when, it is important to remember that this must be translated into real world instrument sounds for the system to work at all. Moreover, the more realistic these instruments sound, the more believable the system will appear to users. There are several existing methods for synthesising instruments, and we will look at two main areas in which they exist; physical modelling synthesis and sample based synthesis.

2.7.1 Physical Modelling

Physical Modelling is a way of synthesising sound by modelling the physics of a real world instrument with a series of equations and algorithms. The first real breakthrough in this field came in 1983, when Kevin Karplus and Alex Strong at Stanford University tried to find ways of generating interesting instrument sounds. This was done by taking a delay line, filling it with random numbers and circulating it. By inverting the signal and passing it back through another delay line we achieve a model of a wave being travelling up and down a string. A filter is placed in it so that the sound decays, and we go round and round, outputting the sound each time. We hear a pitch relative to the length of the delay line.

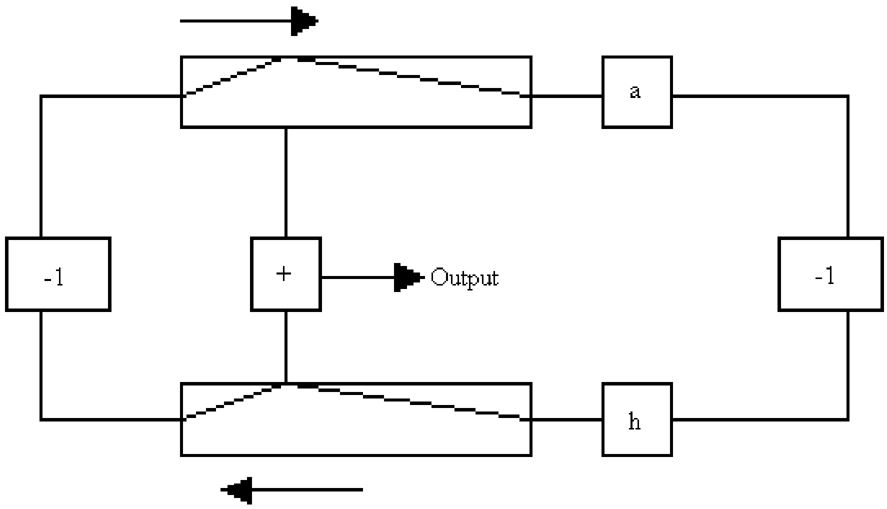
A string in tension has forces in two directions along its length, which will resolve to one force trying to return it to the original position

$\frac{\delta^2 y}{\delta t^2} = c^2 \frac{\delta y}{\delta x^2}$ is the 2D wave equation where c is a constant (property of the string).

f(x+ct) is a solution to this equation:

As time progresses, the shape of the curve created by a displaced string doesn't change, but it moves along the x axis; (*g(x+ct)*) moves left to right, (*f(x+ct)*) moves right to left.

This process can therefore be represented using delay lines:



The signal is inverted before passing through each of the delay lines to model the reflection of the wave at the end of the string. The output is taken from the sum of both delay lines, which will at any stage represent the shape of the string. ‘h’ is a low-pass filter used to decay the sound over time. Without this the note would continue to ring forever. A low-pass filter is used as high frequencies tend to die away faster in the real world, and a low-pass filter achieves this effect.

The pitch that we hear is the length of the delay line multiplied by two (we have two delay lines) plus half a sample, as ‘h’ will add half a sample delay. An all-pass filter ‘a’ also adds a fractional delay, and is used as a ‘tuning filter’ (Karplus and Strong, 1983).

Further progressions of this technique have since been developed to model such additional effects as the frets, the body resonance, and the other strings on a stringed instrument. With application of the right series of filters very accurate models of instruments can be built from these basic principles.

2.7.2 Sample-Based Synthesis

Rather than attempting to model the waveform produced by an instrument, actual recorded samples of that instrument can be used. The issue then is how to produce a full range of note lengths and pitches with only a finite set of samples. A sample-based synthesiser's ability to accurately reproduce a natural instrument’s sound depends strongly on its library of samples. In the earlier days of sample-based synthesis, computer memory was expensive and samples had to be as short and as few as possible.

Looping a part of the sample, and then using a volume envelope curve to make the sound fade away at the end, allowed notes of different durations to be produced. The sample could be played back at a faster or slower speed to change its pitch, and filtered to adjust its amplitude. This technique did not necessarily reproduce a perfectly accurate sound of that instrument playing at different pitches or volumes though, in fact subtle nuances in real-world instruments mean that the waveforms produced will be actually be slightly differently shaped under these circumstances.

As memory became cheaper, it became possible to use multisampling; instead of a single recording, the original instrument could be sampled at regular pitch intervals. This provides a more realistic and natural progression from the lower to the higher notes. It is also possible to sample the same note at several different levels. Amplifying and blending the samples can then make every possible volume in between (Horner, 1993).

2.7.3 Conclusion

Although it would be preferable to generate realistic sounding instruments for the accompaniment, we must bear in mind that this is not the main focus of this project, and there will be time constraints to consider.

CSound already has a ‘pluck’ function, an implementation of the previously described Karplus-Strong physical string modelling technique. This function has been used successfully to produce both stringed instruments and drum sounds. This should cover the range of instruments that will be required for the accompaniment, and by using this existing functionality valuable development time will be saved.

It is therefore intended to initially use the ‘pluck’ function to generate the accompaniment instruments. If the results are not adequate, or if there is further development time in hand, we can look at implementing more advanced physical modelling techniques, and possibly also at sampled synthesis techniques.

2.8 Program Structure

The most interesting and challenging part of this project will be designing and coding the artificial intelligence whose job it is to play the correct chords at the correct time to accompany the musician. In researching other computer musical accompanists, we have found very few other systems that work entirely from the input audio without having a pre-defined score to guide them. It is not, however, an entirely un-researched field, and the principles from other areas of artificial intelligence could also be applicable to this system.

The field of computer intelligence that this system fits in with best is that of Agents. Intelligent Agents and MultiAgent systems are reactive, proactive, social computer systems (Wooldridge, 2002). This system will fit these criteria:

Reactive – An intelligent agent is able to perceive its environment, and respond to changes in such a way as to satisfy its design objectives. In the case of this accompanist system this is true, as it will have to detect the notes that the musician is playing and react accordingly to them.

Proactive – Intelligent agents are able to take the initiative in order to satisfy their goal. Again this is true in the case of this accompanist, there will be no decision making from the user as to what the accompanist plays, the system must listen to the lead musician and actively make its own decision about what to play and when.

Social Ability – Intelligent Agents are capable of interacting with other agents or humans in order to satisfy their design objectives. Interacting with the human musician is obviously fundamental to the success of this accompanist program, and if we model separate parts of the accompanist's abilities as separate agents then interaction between them will also be fundamental.

So this will in fact be an Agent system. There are various approaches recognised in the field of agent design, which may prove useful and relevant to this system.

'Deductive Reasoning Agents' are regarded as the traditional approach to designing artificially intelligent systems. They rely on a symbolic representation of the environment and attempt to reach a desired goal by 'logical deduction' or 'theorem proving' to reach a desired state.

It could be seen as possible to represent the audio input from the musician symbolically, and to attempt to prove theorems stating that the output accompaniment should be in time and tune. Such an approach immediately seems over complicated, and something that would be interesting in theory, but cumbersome and impractical in real-world use. The logical deduction would also be complicated further by the fact that it is a real time system, and so the desired state that the Agent is trying to reach is constantly changing. For these reasons this approach to designing our system appears to be inappropriate.

The 'Practical Reasoning Agent' approach to decision making holds closer to the way in which we would consider a human solving a problem. Here the Agent system holds a set of beliefs, and derives its intentions through deliberation. For instance if this musical accompanist system believes that it is behind time, it will deliberate and decide that it intends to speed up. This model appears to be far better suited to the system that we intent to develop here. The idea of basing the accompanist on a model of a human behaviour rather than theory appeals greatly, as it is human behaviour that we are fundamentally attempting to replicate.

3 Requirements and Design

In this section we will discuss initial design possibilities for constructing the accompanist system. As this is largely an unexplored field, much of the detail of the solution will be developed at the time of coding, and will be discussed at further length in the 'Implementation' section. We will, however, need to enter the coding stage with overall ideas about the structure of the system and how each area will be approached. It is these ideas that will be discussed in this section.

We will also gather and discuss the initial requirements of the program; what at minimum we hope the accompanist will be able to do, and what it should do ideally if we have enough time.

3.1 Human Accompanist Analysis

The system's overall aim is to replace or emulate a small twelve-bar blues accompaniment section. We can therefore draw much of what we want it to achieve by looking at what would be expected from these human blues accompanists if they were following a lead musician.

The first consideration is which instruments we want to have in the accompanying 'band'. A basic blues band will typically consist of:

- Lead Instrument (Acoustic/Electric Guitar, Piano, Mouth Organ etc)
- Rhythm Guitar
- Bass Guitar
- Drum Kit

The musician using the system will play the lead instrument. We can therefore assume that the system itself will be responsible for playing the rhythm guitar, bass guitar and drum instruments. It is the way that these instruments are played that we will need to analyse.

Certain elements of twelve-bar blues are relevant to all the musicians in the band. The song is divided into parts, bars, and beats. There are four beats to each bar and twelve bars to each part. Each of the twelve bars will be played with a backing chord relative to the key of the song, in the order of:

`I, I, I, I, IV, IV, I, I, V, IV, I, I.`

These roman numerals represent the position in the scale of the chord to be played. The first four bars are played in the first (I) position, the root note of the scale itself. If the song is in the key of E then for the first four bars the E chord itself will be played. The fifth and sixth bars are played in the fourth (IV) position. If the song is in the key of E then an A chord will be played, and so on.

There are skills that will be relevant only to a musician playing a particular instrument. It is therefore important that we break the tasks performed by each musician down, and look at the processes they will go through when accompanying a blues lead musician:

Drummer:

- Listen for lead musician’s count-in.

- Wait for the first four beats and then come in exactly on the next (the first beat of the next bar), matching the tempo of the count in.

- Count through the four beats in each bar and the twelve bars in each part following this.

- Constantly listen to the lead musician and adjust tempo to match theirs if necessary.

- On the first and third beats of each bar play the kick drum (In blues drumming it is often popular to play it twice in quick succession on the third beat).

- On the second and fourth beat of each bar play the snare drum.

- On each half beat play the hi-hat symbol.

- At the end of each twelve bar part decide if the song is going to end. If it is then play some final fill and stop, otherwise continue as before.

Bassist:

- Listen for lead musician’s count-in.

- Wait for the first four beats and then begin a blues bass line exactly on the next beat (the first beat of the next bar). This will be played in the pentatonic (blues) scale, and will start on the root note of the scale that the lead musician is playing in. The tempo of the count in must be matched.

- From then on count through the four beats in each bar and the twelve bars in each part.

- Constantly listen to the lead musician and adjust tempo and pitch to match theirs if necessary.

- At the start of relevant bars shift the key of this bass line in relation to the root note according to the twelve bar blues chord changes: I, I, I, I, IV, IV, I, I, V, IV, I, I.

- At the end of each twelve bar part decide if the song is going to end. If it is then play a final root note and stop, otherwise continue as before.

Rhythm Guitarist:

- Listen for lead musician's count-in.

- Wait for the first four beats and then strum the chord relating to the root note of the scale that the lead musician is playing, matching the tempo of the count in.

- From then on count through the four beats in each bar and the twelve bars in each part.

- Constantly listen to the lead musician and adjust tempo and pitch to match theirs if necessary.

- At the start of relevant bars shift the key of this bass line in relation to the root note according to the twelve bar blues chord changes: I, I, I, I, IV, IV, I, I, V, IV, I, I.

- At the end of each twelve bar part decide if the song is going to end. If it is then play a final root note and stop, otherwise continue as before.

3.2 System Structure

As previously discussed in the literature survey it seems that approaching the development of the system using some ideas from the Agent field would be appropriate. By splitting the system into what will effectively be individual Agents running alongside each other, we can develop each competency of the system individually.

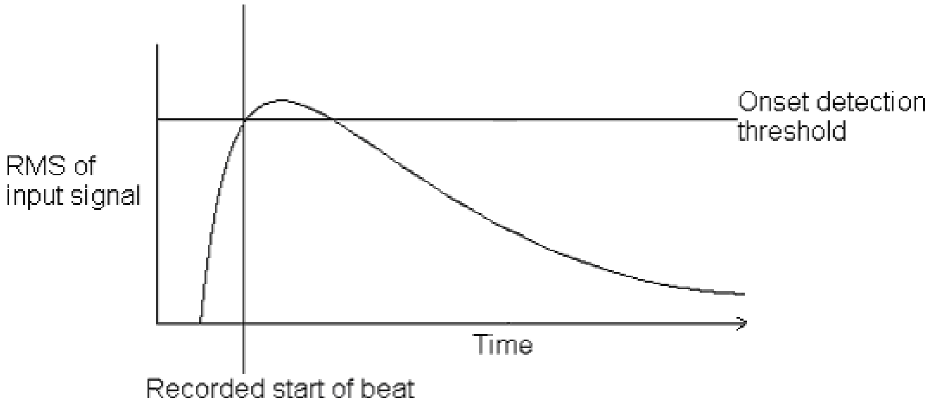
This incremental approach to the programming should make error tracing easier, and will allow us to individually assess how to model human like behaviour in each area. In the sections below we will begin to decide how we will split the program into these separate competencies, and how we will go about developing each one.

3.3 Beat Tracking Competency

3.3.1 Chosen Beat Tracking Method

As discussed in the literature survey, it seems most appropriate to use a very simple onset detector, working by extracting the tempo from the rate of notes played. An onset will be recorded, as the start of a note, at the time the root mean squared (RMS) of the input audio reaches a level above a certain threshold.

The input to the system is likely to be more melodic than percussive, and this method depends on the instrument having peaks in amplitude when a note is first played. If we pick the right threshold level for onset detection this issue should not be critical.



Once notes can successfully be recognised, calculating the beat length, and therefore the speed of the song, is trivial:

```
BeatLength = CurrentBeatTime - LastBeatTime
```

In terms of performance this method requires little calculation, and as we will consistently be listening for the beats throughout the song this is an essential attribute.

3.3.2 Starting the Song

As with a human band of accompanists, we would like the system to be triggered into starting by the musician playing a four beat count-in. This count-in is often given in the real world verbally by counting aloud to the rest of the band. It is also common for the lead musician to count-in by playing four short notes on his instrument, and for simplicity in this system we will assume that this will be the case.

By using the onset detection method to register the beats in this count-in, we should be able to calculate the beat length of the song that is about to start, and come in both at the right time and at the right speed.

3.3.3 Staying in time

It would also be preferable to continue tracking the speed that the musician is playing at throughout the song. The system should then be able to continually calculate the tempo, and react accordingly to it. This will allow the system both to correct any error in its initial tempo belief, and to stay in time with the musician should they speed up or slow down.

3.4 Pitch Detection Competency

3.4.1 Chosen Pitch Detection Method

After an analysis of known methods for pitch detection in the literature review we decided to use the ‘average magnitude difference function’ (AMDF) approach. This was not only because it was felt appropriate in terms of performance, but because an implementation of it is already written into Csound. Although, therefore, we do not intend to implement the method ourselves, it is worth fully understanding it, both for completeness of the documentation, and so that we may make best use of the pre-written function.

This method looks at the correlation between the waveform and the same waveform after a certain ‘lag’. When this lag of a periodic waveform is the length of half a cycle the correlation will reach a minimum, as the delayed wave and the original wave will be out of phase. As the lag increases to the length of an entire waveform, the correlation will reach a maximum, as the two waves will be in phase. This first maximum signifies the period of the waveform.

Rather than looking directly at the differences between the original waveform and the delayed ones this method looks at the average of the square roots of the differences. As a square root is a non-linear compressor this gives a sharpened result, increasing the contrast where it is most needed. This helps with reducing the effect of noise etc.

For each waveform section or ‘frame’, which in the case of the CSound function will be the samples encompassed in one control period, the short-term difference function AMDF is defined as follows:

$$AMDF_n(j) = \frac{1}{N}\sum_{i=1}^{N} \left| x_n(i) - x_n(i+j) \right|, 1 \le j \le MAXLAG$$

MAXLAG is the maximum number of AMDF values generated in each frame. For each frame, the lag for which the AMDF is a global minimum is a strong candidate for the pitch period of that frame. (Ying, Jamieson, and Michell, 1996).

The higher the MAXLAG, the more computationally expensive the algorithm is. If we know the range of pitches that we can expect the note to be in, then this can be kept to a realistic minimum. The ‘pitchAMDF’ CSound function can optionally take values for the range of pitches that can be expected, and we will have to experiment with these values to get the best performance.

3.4.2 Discrete or Indiscrete Pitches

An important choice we must make in the design of the pitch recognition competency is whether to assume that the instrument is tuned to concert pitch or not. Concert pitch is the internationally recognised set of frequencies for musical notes, and any tuning device such as a tuning fork or an electronic tuner will guide a musician to tune their instrument to play these pitches.

Not all musicians will tune to this pitch however, and it is perfectly acceptable to tune to some arbitrary set of frequencies as long as the instrument is in tune with itself and any other instruments that it will be playing with. In the case of human accompany this would mean that the rest of the backing band would have to re-tune their instruments to match that of the lead musicians.

If we assume that the instrument is tuned to concert pitch, then deciding which note the musician is playing will be a matter of choosing the closest matching musical note. This will have the advantage that as long as the musician *is* tuned to concert pitch, the pitch correction will have a relatively large acceptable error margin. As long as the measured frequency is accurate to the nearest semi-tone then the system will be perfectly in tune.

If, however, the pitch correction algorithm is accurate enough to recognise arbitrary frequencies well enough to play acceptably in tune, then the system will be much more versatile. It will mean that the lead musician will not have to correct their to tuning to concert pitch should they not wish to do so, and the system will be able to play along with all instruments tuned to any frequency.

It might initially seem that it will be more complicated to have to re-calculate the blues scales and chord sequences for a completely arbitrary root frequency, but it is worth noting here that from a scientific point of view these are really just ratios from the frequency of the root note. The fifth note in the scale has a frequency of 1.5 times the root note for example. One could, therefore, just as well write the blues chord sequence as a series of ratios from the root note as a series of musical notes:

```
I, I, I, I, IV, IV, I, I, V, IV, I, I.
```

or

```
E, E, E, E, A, A, E, E, B, A, E, E.
```

are equivalent to:

```
1, 1, 1, 1, 1.33, 1.33, 1, 1, 1.5, 1.33, 1, 1.
```

It is in areas such this that there is potential for the system not only to emulate human accompaniment, but actually to improve upon it. Any group of human accompanists will take a few minutes to get in tune if asked to re-tune to match another instrument, but this computer accompaniment system could potentially play to the tuning of any instrument immediately.

As this is a research project it is important to push areas like this as far as possible, to at least show potential for what could be achieved in the field. It is therefore intended that we treat pitch as constantly variable and derive a new scale based on frequency ratios of the recognised root note every time that the system is run. This approach will rely heavily on a very accurate pitch detection competency, but this is a risk worth taking.

3.4.3 When to Check Pitch

In questioning when we should check for the key or pitch that the song will be in, it is useful to look again at the human accompanists that we are attempting to emulate. When human accompanists are expected to join the lead musician immediately at the beginning of the song, it is essential that they already know they key.

The lead musician will often convey the key verbally to the rest of the band just before starting the song. Occasionally however, the lead musician will simply count the rest of the band in using the root note to mark the beats, therefore indicating both the key and tempo of the song. Other musicians will then be expected to either recognise the pitch, or more often (in the case of guitar at least) to simply look at the position that the lead musicians are on their instrument.

This method would appear to be particularly appropriate to this system, as it maintains the principal of no interaction between the musician and the system other than through audio, and also allows for the musician to play in any arbitrary pitch as discussed in the previous section. We will therefore assume that the note used to play the count-in beats will be in the key of the song, and it is here that we will check for pitch.

3.5 Ending

Knowing when an improvised song is going to finish is amongst the most abstract skills that a human accompanist will have. The only initial knowledge that they have is that the song is very likely to finish at the end of a twelve-bar part. It seems very hard to define the ‘feeling’ that the song is about to finish, although this is definitely something that is present even in people who are not musicians. It can sometimes be a change in the way the notes flow, sometimes a change in tempo, sometimes a change in volume, or a combination of any of these. This is certainly a part of music that is difficult to analyse scientifically. It will be a challenge to attempt to model this skill.

The feasible answer in this system may in fact simply be to wait until the lead musician has stopped playing, and for the accompanist to stop at a reasonable point following that. This is a technique that would perhaps be expected of a much less accomplished human musician. It would have to be judged carefully so as neither to continue for too long or to accidentally stop early. We will experiment with these different ideas at the time of implementation.

3.6 System Requirements

As this is a research project the initial requirements will be less detailed and less constrictive than an in depth set of requirements such as those that might be needed for a traditional software engineering project, and are more geared towards overall aims. Before implementing an experimental system such as this one, it is often difficult to foresee how far it will be possible to develop its features. The requirements are therefore separated into minimal requirements that are fundamentally needed for the system to be considered successful, and some further optional ones that it would be nice to implement.

3.6.1 Basic Requirements

- Start the song correctly along with the lead musician.
    - Start at the right time.
    - Start at the right tempo.
    - Start in the right key, ascertaining the correct pitch whether the lead musician’s instrument is tuned to concert pitch or not.
- Play at least the minimal accompanying instruments.
    - A simple drum or percussive sound at the correct time and speed to maintain rhythm.
    - A bass sound playing just the root note at the correct pitch.
- Change key according to the twelve bar blues chord sequence at the correct times.
- The system should require no information from the lead musician other than through the audio input from their instrument.

3.6.2 Further Requirements

- Stay in time throughout the song.
    - Correct any error in the current assumption of tempo and/or beat position.
    - Update the current assumption if the lead musician should choose to speed up or slow down.
- Stay in tune throughout the song
    - Correct any error in the current assumption of the pitch that the song is being played in.
    - Update the assumption should the lead musician change key.
- Provide a realistic sounding full blues backing band
    - A full drum kit with kick drum, snare, toms and symbols, playing fills at appropriate times.

  - A bass guitar that plays a blues bass line rather than just the root note.

  - A rhythm guitar that plays the correct chords and blues riffs.

  - Further instruments such as brass or keyboards/piano.

- Be able to recognise when the musician is playing a less common blues styles such as eight-bar, and change chords appropriately.

- End at an appropriate time.

  - End if the musician stops playing for a reasonable period of time

  - End with the musician if they change tempo or play notes in such a way as to indicate that they are approaching the ending.

4 Implementation

In this section the actual development of the system will be described, covering things that worked and things that did not, ongoing changes that had to be made. Any decisions taken during this process will be justified.

It was important, as this was largely an experimental prototype system, to remain open minded to new ideas, or even to major design changes during the implementation, were they required. This led to an ongoing cycle of coding, testing, and (re)designing throughout the implementation of the system.

During the development guitar was used as the main lead musician input. Guitar types would regularly be switched between electric and acoustic. Occasionally piano was also used (although the authors skills here were limited). It was important to achieve a balance between having the ability to play like a lead blues musician and maintaining the system’s versatility.

4.1 Analysing real-time Input

In order to begin programming the useful parts of the system, some groundwork needed to be done. It is important here to remember the way in which CSound samples and executes sound:

Sample Rate – The number of samples taken of the wave per second. This has an important bearing on the system, as performance is so vital. Using a higher sampling rate is likely to result in more accurate input analysis and higher output sound quality. Using a lower sampling rate is likely to result in less processing time, and therefore better real-time performance. A balance is obviously required.

Control Rate - The number of times that code is executed per second. It is vital to pick a reasonable value here for several reasons. Sound output will be triggered from these code execution points, and it is almost certain that the beats in the song will not be exactly aligned with them, so they need to be regular enough to make any offset with the beats in the song negligible. Another important consideration is that the control rate will result in a ‘samples per control period’ ratio. The number of samples in each control period is very important in this system, as we will be analysing these chunks of samples for pitch and beat extraction, and so they need to contain a certain amount of samples for good results.

Now although the system needs to run in real-time, the idea of detecting a beat and playing along with it at exactly the same time is impossible. We not only need to allow time for each set of samples to be collected in each control period, but also for analysis to be made on these sample sets. The time this takes will be referred to as the latency.

We can overcome the effect of latency by predicting when the next beat will occur rather than attempting to play along with the current one, so that we are ready in advance. Obviously if the latency ever becomes greater than the period between beats then the problem becomes exacerbated, it is imperative that we avoid this. The sample and control rates will play an important role in keeping the latency below this threshold.

Appropriate values were found using a combination of looking at previous real-time CSound systems, and through experimentation. Like many other parts of the system they were experimented with and re-evaluated throughout the development process. The final values settled on were:

```
Sample Rate: 10,000
Control Rate: 125
Samples per Control Period: 80
```

Another point about calibration here is that performance was greatly increased by using fewer sound channels. It was not felt that stereo sound would be of any particular benefit, especially as this system is essentially only an experimental prototype, and so a single channel (mono) system was decided upon.

4.2 Initial Beat Tracking

It is worth defining some terms relating to tempo that will be used to describe the system:

Bar – A subdivision of a song. Blues music (for the purposes of this system at least) consists of distinct parts made up of twelve bars with repetitive chord changes.

Beat – There are four beats to a bar in most blues music, and hence four beats in the count in. Often notes that the musician plays will be exactly on the beat.

Tempo – A measure of the speed of the song, which is usually given by musicians in beats per minute (BPM). Within this system it is far more useful to know the speed of the song in terms of ‘beat length’ however.

Beat Length – This can also be seen as the time between beats. It is a measure of the speed of the song. By storing the length of a beat rather than a BPM value, the system could predict when the next beat would occur without any unit conversions.

Beat Position – This is distinctly different from, yet equally as important as the beat length. The system needed to know not only the speed of the song, but also the position in time at which the beats occurred.

RMS – The ‘root mean square’ of the audio input was used a measure of its loudness at a given point. This was useful in onset detection, as will be described in more detail in the sections below.

It was decided to implement an independent competency to analyse the count-in played by the musician. Its job was to recognise the position of the beats in the count-in and to provide tempo information to the rest of the system in time for the information to be useful. This was a critical part of the program. If the accompanist could come in on time and be playing at the right speed, then it would begin to show the characteristics of human accompaniment.

4.2.1 Onset Detection

As discussed in the design section the basic method for beat detection will use very basic use onset detection – evaluation the root mean squared (RMS) value of the input signal. The values of the samples in the given control period are squared and rooted, simply so that negative values become positive. This then gives larger values for higher amplitudes. The mean of all these values for each of the samples in the control period then gives the RMS value. Once this value becomes higher than a certain threshold we will assume that the musician has played a note at that point, and record the time as one beat of the count-in.

4.2.2 Threshold Level

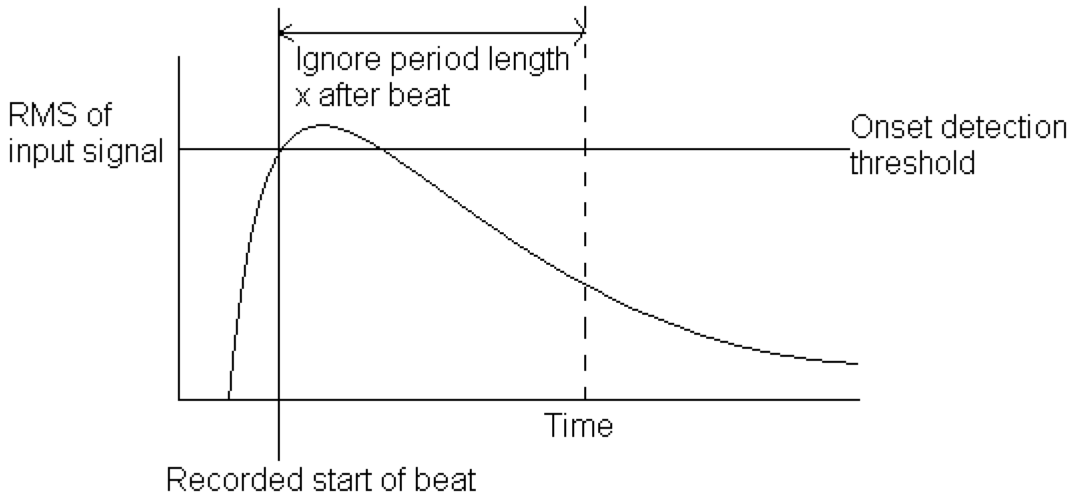
It is important that the value of the threshold level for the RMS in the onset detection is high enough that is does not record any background noise as beat in the count-in. It must also be low enough that it does register the musician playing at the quietest level that they might normally play. This may change for different musicians and different instruments, and was set as one of the calibration variables clearly marked at the top of the program code.

4.2.3 Ignore Period

One problem of using this method during the count in, is that the RMS of the input signal will almost certainly remain above the threshold level for a length of time while the note rings. This is not another note being played, it is simply the remainder of the note that has already been registered, and so we need to ensure that during this time multiple beats are not falsely recognised.

This problem was solved by ignoring the input signal for a set time period after each of the count-in beats was recorded. This period could be calculated reasonably, as we knew the fastest reasonable tempo that a musician would play, and hence the shortest possible time in between beats.

A diagram illustrating the onset detection method described is shown below:



4.2.4 Tempo calculation

Once we can recognise the time location of the count in beats we can calculate the tempo of the backing music to play. Blues is nearly always played in a 4/4 time signature (4 beats per bar), and so for the purposes of this system we will assume that the count in is four beats long. If we start a timer at the first count, and stop it at the fourth, we will have a period of three beat lengths. We can therefore simply divide this time by three to find the beat length:

```
BeatLength = CountPeriod / 3
```

4.2.5 Compensating for Latency

After the last beat in the count-in, the average length of a beat was calculated, using the method described above. This provided information about the tempo, and latency in the system had no profound effect on this. It was in the determining of the beat positions that latency came into play – it is one thing to play at the right speed, but to play in time requires beats to be synchronised exactly. Our basic method for working out the position of the first beat would initially be to wait for one beat length after the last count of the count in:

```
BeatPosition = EndOfCountTime + BeatLength
```

As noted however, there would inevitably be delays in the system from the processing of the input to the generation of the output. A latency constant was used to allow for this delay, and simply subtracted from the calculated beat position:

```
BeatPosition = EndOfCountTime + BeatLength - Latency
```

The value of this latency constant was calculated mainly through trial and improvement, it was fairly easy to judge when the accompanist started too early or too late. It is worth noting here that computer systems with different hardware, operating systems, and compiler versions etc will almost certainly require this constant to be recalculated, and it was set by another of the system calibration variables at the top of the program code.

4.2.6 Algorithm Pseudo Code

Here is the pseudo code for the method of analysing the count-in described above:

```
For Each Control Period
 If(InputRMS > Threshold AND CurrTime > IgnorePeriod)
  Count = Count + 1
  IgnorePeriod = CurrTime + IgnoreLength
  If(Count == 1)
   FirstCount = CurrTime
  EndIf
  If(Count == 4)
   LastCount = CurrTime
   BeatLength = (LastCount – FirstCount)/3
   BeatPosition = LastCount + BeatLength - Latency
  EndIf
 EndIf
EndFor
```

4.3 Ongoing Tempo Adjustment

The next consideration in terms of timing was adjusting the tempo to keep in time with the musician as the song progressed after the count in. Two distinctly different approaches to solving this problem were developed.

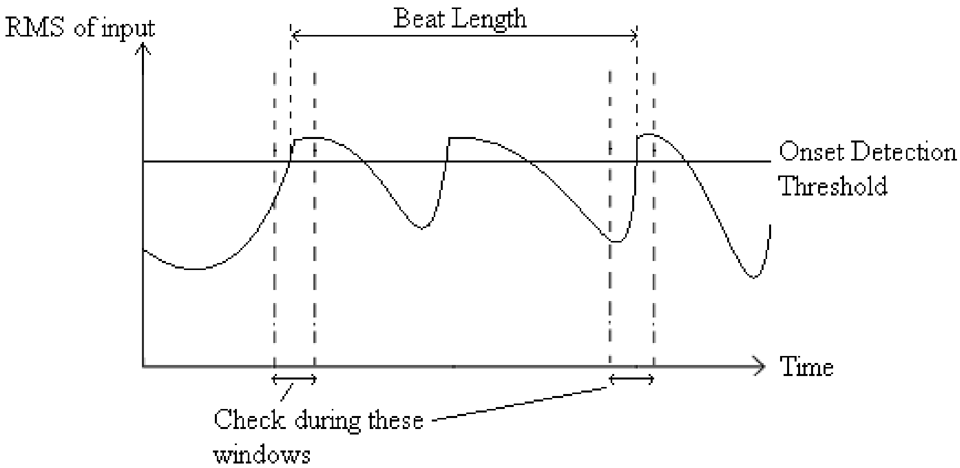
The first one, in a similar way that the method for timing the count-in did, measured the distances between the notes that the musician played and calculated the tempo from that. The second predicted where a note might occur, and then measured the difference between the prediction and the reality to find out if the system's tempo belief needed adjusting. Both methods are described in detail below.

4.3.1 Method 1

The first method of ongoing tempo recognition and correction that was implemented was one that measured the distance in time between the notes that the musician played. An important consideration here was that we did not want to measure any note that the musician played, only the ones that they intended to be played on the beats. It was also important to realise that the musician would not necessarily play a note on the beat at all.

Once we could separate out only those notes intended to be played on the beat, and once the musician played two of these in a row, then we could measure the time between them. We could then compare the system's current belief about the beat length with this value, and update it accordingly.

The method used to recognise only those notes intended to be played on the beat was to use windows around the times when beats were expected to occur, and only to check for onsets within these windows. The size of these windows was of course critical, too big and they would pick up notes not intended to be played on the beats, too small and they might miss ones that were. This method is represented in the diagram below:



If the RMS of the input was already above the threshold level as the window was entered then the result was discarded. This was because it was likely to be a note that was played previously and was still ringing. It is only the exact start of notes within the window that interests us.

This is the pseudo code for the algorithm that will be repeatedly called during the song at the control rate to implement this method:

```
If (Time = WindowStart AND InputRMS > Threshold) //already ringing
 GotFirst = False
 STOP CHECKING TILL NEXT WINDOW
ElseIf (WindowStart < Time < WindowEnd AND InputRMS > Threshold)
 If(GotFirst)
  SecondBeat = Time
  BelievedBeatLength = SecondBeat – FirstBeat
  FirstBeat = SecondBeat
  STOP CHECKING TILL NEXT WINDOW
 Else
  FirstBeat = Time
  GotFirst = True
  STOP CHECKING TILL NEXT WINDOW
 EndIf
ElseIf (Time = WindowEnd)
 GotFirst = False
EndIf
```

4.3.2 Method 2

After some testing of the previously described method it was felt worthwhile to experiment with a completely different method of tempo comparison and adjustment. This time the distance in time between an individual note that the musician played and the time that we expected them to play it was measured.

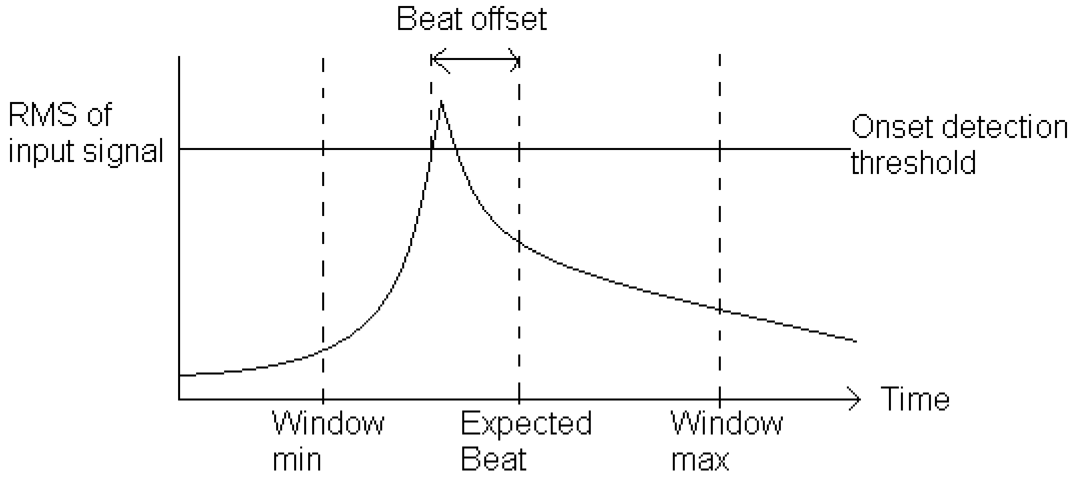
Again we only wanted to deal with notes that the musician intended to play on the beat, and so again used a window around the expected time to cut out other notes. Any results when the input RMS was already above the threshold level at the start of the window were discarded. As before this was because it was likely to be a previously played note still ringing.

If a note was played within this window, then the time at which the note was played could be compared with the time that was predicted, and this difference used to calculate whether the system was playing too fast or too slow. The system's belief about beat length could then be updated accordingly. This is the pseudo code for the algorithm that was repeatedly called at the control rate to implement this second method:

```
If (Time = WindowStart AND InputRMS > Threshold) //already ringing
 STOP CHECKING TILL NEXT WINDOW
ElseIf (WindowStart < Time < WindowEnd AND InputRMS > Threshold)
 RealBeatTime = Time
 TempoError = RealBeatTime – BelievedBeatTime
 STOP CHECKING TILL NEXT WINDOW
EndIf
```

This algorithm provided the tempo error, which was either a small positive or small negative time value. It, or some fraction of it, was then added to the believed beat length value to update it accordingly. The development of this part of the process is described in more detail in the 'Reaction to Tempo Change' section.

The diagram below represents this second method:



4.3.3 Method Comparison

A very important issue not addressed by the theoretical algorithms, is the practical implications of latency caused by input and output delays and the processing time of the computer. When dealing with the timing of a song even a tiny misjudgement in the latency compensation will lead to a noticeable error.

In analysing the results of these two separate methods of tempo correction the vital difference regarding the latency problem became clear. The first method does not rely on an exact prediction of the beat position with latency accounted for, just a rough one accurate enough to get the window in the right place. The second method however relies on an exact prediction of when to expect the note to be played, accounting exactly for the latency of input, output and processing.

This means that by comparing the input to itself in the first method the problem of latency compensation is removed, and you would expect more accurate results. In the second method any error at all in allowing for latency will make the system believe it is out of time by an incorrect amount.

While this is true, and initially the first method did produce more accurate timings, the very fact that the input was only compared to itself produced another far more serious problem. Because there was no association with the backing music that the accompanist system was playing, the method was effectively only checking for error in the beat length i.e. the tempo, and not the beat positions. This meant that, if the musician and the system were playing at exactly the same speed, then the system would believe that it was in time, even if it was out of synchronisation.

What was needed was a method that measured for both beat length and beat position variations, and the second method did exactly that. If the musician sped up then notes would be registered before predicted, and the believed beat length would be reduced until the accompanist was in time again. As well as this, if the speed was right but the believed beat position was too late, then this difference would also be recognised. Because the musician would be playing the notes before the predicted time, the accompanist would in fact speed up until it was back in synchronisation, and then slow back down to the musician's tempo. This is just what a human musician would do, and it is exactly what we want.

For this reason it was decided to accept that latency compensation was going to be far more critical, and that development should continue using the second method.

4.3.4 Reaction to Tempo Change

We have only so far described methods of recognising tempo changes or errors, and not gone into any detail about how the system should react when they occur. This is a very interesting area, and provides characteristics that have a great bearing on how human the system will sound. Several methods were used throughout the development of the system, each one improving on the last as understanding was built.

Initially the tempo error was simply added on to the believed beat length after it was calculated:

```
BelievedBeatLength = BelievedBeatLength + TempoError
```

While this method worked in as much as it did change speed when the lead musician did, the tempo changes were too harsh and immediate to sound human. There was an unnatural jumping in the backing music which broke the flow and feel of the song. After analysing some human accompanists it was realised that as a lead musician speeds up, the accompanying musicians will not instantaneously react and adjust exactly to the new tempo, but rather will do so gradually. The initial attempt to model this effect was to use a constant fraction to multiply by the tempo error by when adding it on (a value of around 0.3 worked best):

```
BelievedBeatLength = BelievedBeatLength + (FractionChange *
TempoError)
```

This meant that the accompanist system approached the new tempo gradually over a few beats rather than all in one go. This sounded much more realistic with the song continuing to flow far more smoothly. There still seemed to be an absence of a real human sense of rhythm though, and after further analysis of human accompaniment the difference was found.

Human accompanists do not directly alter the tempo they are playing at, but rather, in a manner of speaking, adjust their *acceleration*. That is to say that they alter their rate of tempo change. This means that they will change tempo faster if the tempo has already been changing for the previous few bars, and they will continue to change until they approach the lead musician's new tempo where they will gradually reduce the rate of tempo change until they again match him exactly. By doing this, the song never loses its rhythm and feel, rapid tempo change can occur, but it must be built up to.

This meant that we needed some memory of what the tempo had been for the past few bars, or even for the whole song. An attempt at a method using arrays to memorise past tempo changes, but it was soon realised there was a much simpler solution. What we needed was to add the tempo error not directly to the believed beat length, but rather to a separate variable storing overall tempo change:

```
TempoChange = TempoChange + (FractionChange * TempoError)
```

Then add this instead to the believed beat length every beat:

```
BelievedBeatLength = BelievedBeatLength + TempoChange
```

In this way the tempo change variable in effect represented the rate of tempo change, or acceleration, and just like with a human musician it was this that was adjusted rather than the tempo directly. Another observation about human accompaniment was that tempo changes for the most recent few beats have far more influence than tempo changes from several beats beforehand.

With this method so far, tempo changes from the entire song had equal bearing, which was obviously not right. What we needed was to multiply the tempo change variable by a constant fraction each beat - in effect a simple filter.

```
TempoChange = TempoChange * TempoFilter
```

After some experimentation it was found that halving the variable each beat produced great results, and the accompanist now changed speed smoothly, and sounded like it had far more realistic and human sense of rhythm.

4.4 Initial Pitch Detection

It is first worth defining some terms relating to pitch that will be used to describe this part of the system:

Scale – The set of notes that both the melody and the chords of the song will be made up from. The blues scale is based on a minor pentatonic (five notes per octave) scale which is defined by I, iii, IV, V, vii (root, minor third, perfect fourth, perfect fifth, minor seventh) relative to the root note. Thus A minor pentatonic would be A, C, D, E, G. The instantly recognisable blues sound comes from adding in also an augmented fourth, which is referred to as the "blue note". The A blues scale would therefore be A, C, D, D#, E, G, A. A scale can also be seen as a set of pitch ratios from the root note. Seeing as we are not assuming the lead musician will be playing in concert pitch, we will entirely deal with pitches and scales in terms of these ratios.

Key – The key of a song refers to the scale that it is played in.

Root Note – The term ‘root note’ is used to specifically refer to the pitch of the first note in that key’s scale. This can also be referred to as the ‘tonic’. This is the note that the musician will play when counting the accompanist in. This system stored the ‘believed root note’ at all times during the song. This was its current assumption of the root note of the key that the musician was playing in.

Semitone – The smallest musical interval used in most western music. In equal temperament this is a ratio of approximately 1:1.06 between frequencies.

As discussed in the design section, it was felt that the best way of conveying the initial key of the song, i.e. the pitch of the root note, was to use that note for the count in.

The pitch detection function was relatively computationally costly, but this was not too much of a problem, as it did not need to be run at every control period like the RMS function for onset detection. The problem then was choosing when to run it.

At first a single reading was taken on the final note of the count-in, which provided acceptable results. Performance was slightly improved, however, when a reading was taken on all four of the notes in the count-in and average between them taken. The pseudo code describing this method is shown below:

```
For Each Detected Count In Beat
 FrequencyCount = FrequencyCount + CurrentFrequency
 If (FinalCount)
  FinalFrequency = FrequencyCount/4
 EndIf
EndFor
```

4.5 Ongoing Pitch Correction

While developing the system it was found that the system would sometimes play very slightly out of tune after the count-in. This was initially blamed on flaws in the pitch detection method itself. Although it was only by a very small amount, it was noticed that the system was very slightly sharp (too high) when an acoustic guitar was used as an input.

It was finally realised that there was no problem with the pitch detection method itself; the problem was a physical one with the way the guitar was played. When counting in using the root note, short notes were being played, and the strings were being plucked quite hard. When melodies were played during the song, however, plucking was gentler to suit a more flowing style.

A string that is plucked hard will have a higher pitch at first due to the extra tension caused by the string deflecting further. This is noticeable for instance in the ‘twang’ made by a plucked elastic band. It is not usually noticeable in a guitar as it is so short lived. This system however was tuning all the other instruments to the pitch of that initial hit of the string.

What was needed was to initially take the pitch detected during the count-in, but then to refine this belief whenever the musician played another root note during the song. The issue here was to identify when the musician was playing a root note.

The closest note that a musician is likely to play to the root note will be one semitone away either higher or lower. Even this is fairly unlikely as the notes a semitone away are not part of the blues scale. They are occasionally utilised in ascending and descending lead parts however, and so it was important to play safe and assume they would be used. We could therefore say that if we detected a pitch that was less than half a semitone (3%) away from the believed root note, then it is likely that the musician is playing a root note. In this case we will update the believed root note to be the average of the current belief and the note just played. This is the method implemented to fine-tune the pitch during the song:

```
For Every Beat
 If (BelievedRoot*0.97<CurrentPitch<BelievedRoot*1.03)
  BelievedRoot = (BelievedRoot+CurrentPitch)/2
 EndIf
EndFor
```

In this way the system's belief was updated every time that the lead musician played a root note. Any very slight errors throughout the song were gradually reduced.

4.6 Instrument Modelling

It is important to remember that the main focus of this project was on the intelligence behind playing along with the lead musician. Instrument sounds did need to be generated to allow the system to work at all though, and so although not a primary focus they were part of the project nonetheless. This was another element of the system that was gradually improved throughout the implementation process. The models and methods used to generate sounds for each of the instruments are described below.

4.6.1 Guitar

For the guitar an instrument was implemented that modelled a single string of a guitar being plucked. This instrument was then played several times in parallel at various different pitches to generate chords.

The string pluck was modelled using the Karplus-Strong technique as described in the literature review. This was made easier by the fact that the technique was already implemented in the CSound 'pluck' function. By looking at some previous CSound guitar instruments, and by taking ideas from these and experimenting, a combination of the function variables and filters were found that improved upon the basic sound produced.

4.6.2 Bass Guitar

Interestingly, for the bass guitar it was found that a very simple sinusoidal oscillator could be used to produce a slightly synthetic, but acceptable bass guitar sound. When this was used as a regularly tuned guitar it sounded terrible. It seemed as though, to the author's ears at least, when sounds are towards the lower end of our hearing range we become less sensitive to the subtle nuances in wave shape. This was by far the most simple instrument implemented, and yet arguably sounded the closest to the real thing.

Such psychoacoustic effects, while not directly related to the goals of this project, are still fascinating, and could certainly have a bearing on the way in which systems like this one can be designed.

4.6.3 Drums

The Drums are in fact a series of different instruments, as a drum kit has several components that need to be modelled individually.

A basic drum rhythm consisted of alternating kick drum and snare drum beats. It is the kick drum that is played on the beat, and it is very important that this sound is deep and full, as this gives the song its real sense of rhythm. The snare drum needs to be higher pitched sharper sound, so as to keep the drums flowing, but not to overpower the bass drum's fundamental groove.

By looking at previously CSound drum kits, and through experimentation methods were found that helped to create distinguished kick drum and snare drum sounds. These drum sounds were produced by taking combinations of sinusoidal oscillators and randomly generated noise. These were then passed through a series of low-pass and resonant filters and recombined to generate the drum sounds.

4.7 Backing Music Generation

The backing music had to be dynamically generated at each beat, so that tempo and pitch changes could be reacted to within one beat length.

It is worth noting that in a CSound orchestra file any functional code is written within an 'instrument', and these often are actually be instrument models such as those just described. They can just as well, however, be controlling code. In terms of the code structure the first CSound 'instrument' dealt with all the input analysis that has been discussed so far. A separate second one was then created that controlled the backing music generation. In this way the instrument acted as a function that was called at the beginning of each beat to generate the right music for that individual beat. It had access to the global variables set by the first instrument, providing all the information about tempo and pitch that it needed.

4.7.1 Chord Calculation

Each time the music generation function (or instrument) was called, it counted forward through the four beats in each bar, and through the twelve bars in each part. This maintained knowledge of current position in the song was essential when calculating the chords and generating the music.

By knowing which of the twelve bars it was currently playing in, the accompanist was able to calculate when to change chords, and what that chord should be according to the twelve bar blues progression. With knowledge of the frequency of the root note all other frequencies for the following chords could be calculated. Chord changes were then made according to the twelve bar blues chord sequence, as described in the table below:

Bar	**Chord to play**	**Frequency Relationship to Root**
1	I (Root)	Root*1
2	I (Root)	Root*1
3	I (Root)	Root*1
4	I (Root)	Root*1
5	IV (Fourth)	Root*1.333
6	IV (Fourth)	Root*1.333
7	I (Root)	Root*1
8	I (Root)	Root*1
9	V (Fifth)	Root*1.5
10	IV (Fourth)	Root*1.333
11	I (Root)	Root*1
12	I (Root)	Root*1

The system was therefore always aware of the current chord frequency, and the position of the beat in the bar. It would also be aware of the speed of the song as it had access to the constantly updated 'beat length' global variable. This was everything that was needed to generate the music for each instrument for that beat. This was calculated as described for each instrument below.

4.7.2 Bass Guitar

A nice sounding simple blues bass line was played in each bar following the blues scale. It started on the root note of the current chord for the first beat in a bar, and used frequency ratios to work out other notes in the blues scale relative to this note. It then played notes from this scale on the other beats.

4.7.3 Rhythm Guitar

Blues chords are fairly simple, and will often just be fifth chords, made up of the root note, the fifth above this, and the root note an octave higher. Rather than just playing this chord, however, a simple riff was also created by shifting the fifth up to a seventh on the off beats (even beat numbers in the bar). This immediately gave a bluesy feel to the rhythm guitar part.

4.7.4 Drums

For the drums a very simple drum kit was played using just a kick drum (sometimes called a bass drum), a snare, and one hi-hat style muted symbol. A simple bluesy riff is played as follows:

Kick Drum- Play on the first beat of each bar, and twice at half a beat length apart on the third beat.

Snare Drum - Play on the second and fourth beats of each bar.

Hi-Hat Symbol – Play twice at half a beat apart on every beat of the bar.

4.8 Ending the Song

4.8.1 When to End

The final competency to implement in the system was one that recognised when to finish. So how do human musicians recognise when a song is likely to end even if they have not heard it before? It does not appear to be a difficult skill, the most amateur of bands jamming out a new song will nearly always manage to come to a close together.

During a practise with the band that the author currently plays guitar in, an attempt was made to ascertain the processes that musicians go through when listening for an ending. Immediately something was noticed that would be difficult to model in a system such as this one. Hearing was not the only sense used by the band. Often the body language of the musicians, and eye contact made were equally important.

As an exercise the band was asked to try and jam together, and end together, with their eyes shut. Although this was in fact successfully achieved, they were not playing as tightly as before, and a couple of the instruments were a little indecisive at times. Surprisingly it seemed harder to stay in time during the entire song, and the transitions between loud and soft parts were a far greater challenge as well as the ending. Although a little sloppy, however, they did still manage stop together, proving that it would be possible to replace visual communication with heightened listening skills. Some of the greatest blues musicians of all time were completely blind, so this is obviously the case!

One option considered was to model the Lead Musician signalling to the rest of the band. This could have been done with a key press indicating that the song would finish after the next twelve bar part. Getting CSound to recognise a keyboard or mouse input in real-time was proving to be a problem however, and before this competency was even fully implemented it was decided that it was not a suitable solution. The system had thus far required no interaction between the musician and the computer other than through audio. By introducing a key press one of the basic requirements of the system was not being adhered to. On top of this it was found that, without any interaction other than through audio, the whole experience felt more like one was playing with a human musician rather than with a computer. For a musician, having to take their hands off their instrument to press a key felt uncomfortable enough to abandon the idea of using this method.

So the question now was whether we could define the things that human musicians look for in melody, tempo, and volume changes to recognise when the end of a song was being approached. It was not felt that there would be enough time to implement such a competency successfully in this prototype version of the accompanist system, but some ideas about how one might go about it in the future are discussed in the 'Further Work' section.

The remaining option in this system was simply to wait until the lead musician had stopped playing for a sensible amount of time, and for the accompanist to stop at a reasonable point following that. A twelve bar blues song will almost invariably finish at the end of one of the twelve bar parts. The method therefore checked at the end of each part, and stopped if there had been a long enough silence leading up to it:

```
For Each Beat
 If (InputRMS < LevelThreshold)
  SilenceCount = SilenceCount + 1
  If (Bar = 12 AND Beat = 4 AND SilenceCount > SilenceThreshold)
   END THE SONG
  EndIf
 Else
  SilenceCount = 0
 EndIf
EndFor
```

The 'Silence Threshold' had to be judged carefully so as neither to continue for too long or to accidentally stop mid song. After some experimentation it was found that waiting for a silence throughout the last bar (four beats) was reasonable.

4.8.2 How to End

Once we have successfully recognised an appropriate time to end, we must also have an appropriate way to end. At first the system simply stopped. This in fact sounded even weaker than one would already expect, as though somebody had just turned a record off, and was very unnatural. It was considered that maybe the accompanist should fade out gradually with a steadily decreasing volume. As this system was a model of live human accompaniment this would not, however, have sounded at all realistic.

Live blues endings can be very complex and drawn out, and these longer endings are often entirely improvised. One can sometimes feel as if musicians use the ending as an escape from the constraints of the rest of the song and an opportunity to show off their most impressive off the cuff playing. Improvisation like this is obviously not something this system would currently be capable of.

It is in fact perfectly acceptable, however, for an accompanying band to play a far simpler ending. A single drum hit with a sustained root note from the guitar and bass invariably sounds good, and so this method was chosen to be implemented.

5 Evaluation

As the program has now been developed as far as we intended in this project, it is important to look back over the development process, and to analyse how successful the processes and overall project have been. It is more important still to see how independent users react to the system, and to get feedback from them.

5.1 Research

Researching past material for this project was a little difficult. Some areas of what was planned were not previously researched or documented very well, but this was to be expected as the system was exploring some new territory.

The nature of the project was such that it was hard to know quite what would be required in terms of preliminary research. Now that the project has been programmed it seems obvious that there are fields that should have been given more attention at the research stage, and others that were researched needlessly. This is the nature of an experimental or research type project – it is difficult to know exactly what to expect, and with hindsight mistakes are inevitably made.

As most of the research areas were very specific, most useful information was from papers and journals rather than from textbooks. The research done was worthwhile and helpful overall, and the designing and programming of the project was approached from a stronger position because of it.

5.2 Design

Again in the design section it was often difficult to know what the real challenges would be and what would work best until actually experimenting with them during development. Therefore the design section did not go into intricate detail, and many of the design decisions for the system were actually taken at the time of development.

The overall structure of the system was decided though, and different possibilities for how certain competencies would be implemented were looked at. This provided a better overall idea of the order in which the areas within system would be implemented, and of the timescale to adhere to. Having at least a general idea of how each problem was going to be solved was vital too, and if the coding had been approached without this, fundamental issues could have been overlooked, and the timescale lost track of.

By working through the different areas of the system in the design stage, problems that might have arisen were already being considered, and the way that the overall system would have to fit together was envisaged. A lot of understanding was gained in these areas, and despite not designing the system down to the last detail, the coding stage was entered from a far stronger position.

5.3 Program Implementation

Both the nature of the project and the structure of the code suited an incremental style of development. Each of the competencies was relatively independent, which allowed each to be focused on independently. This was useful as each component could then be tested at the lowest level and build upwards from, rather than having to try and distinguish where these faults were coming from later. This made the coding process much more efficient.

The only way to develop and improve many of the competencies in this system was to try them in practice and see how they functioned or what they sounded like. This meant that the program implementation consisted of a mixture of general ideas taken from the design stages, and ongoing experimentation at the time of coding. A process of trial and improvement was required for many of the competencies, and the skills required for this process were built up as the development continued.

In some cases this process could break down altogether and a dead end could be reached. In these cases it was necessary to take a few steps back and attempt a completely new approach to solving the problem, learning from the flaws of the previous effort. Again knowing when this was required was a skill that was built and improved upon greatly as the development went on.

As this was the first real programming the author had ever done in CSound the learning curve was steep. Because the system was so reliant on performance, Areas of the code would often have to be re-written and streamlined as skills were gained. Somebody who had knew CSound to a professional standard could certainly have still found ways improve the code in places.

For development of a prototype, experimental system, where the emphasis was on research rather than producing a polished commercial product, the approach taken was generally the right one, and allowed most of the goals in the development of the system to be met.

5.4 User Evaluation

Now that the system has at least reached the state of a finished prototype it will be very interesting to gauge its reaction with experienced musicians. After all the aim of the system was to mimic human accompanists, and it will be fascinating to see to what extent users believe this has been achieved. It will also be useful to see what users feel are the weak points of the system.

Each musician will be asked to perform a song along with the accompanist system. This will be recorded. Their opinion of the system will then be obtained by asking them a series of questions.

5.4.1 User 1 – Clean Electric Guitar

This user test can be heard on the audio c.d. track 2

Could you tell us which instrument you play, and give us an idea of your experience of playing blues music?

I play lead guitar, and today I'm playing a fender telecaster. I've played blues guitar for over 30 years and used to be in a few local blues bands, nothing too serious.

How successful do you think the accompanist was at coming in on time and in tune?

I was impressed by how well it did come in, it sounded perfectly on time but listening back to the recording it is very slightly out of tune at times. Andrew explained that is because it tries to register the exact pitch of the guitar so that it'll work even with a detuned guitar. As long as the guitar is in tune with itself it'll pick up the right note, I like that because it saves hassle.

Did the accompanist stop at the right time and did its ending sound good?

On the piece I recorded it played the next 12 bars after I finished. Apparently because it waits for you to clearly stop playing and I played slightly into the next 12 bars it finished those off too. The next time I played I finished slightly before the end of the 12 bars so it finished on those. The ending is simple but fine.

How successful do you think the accompanist was at staying in time and tune throughout the song?

During the part I recorded I played at a pretty constant tempo, and the accompanist stayed at the same speed. Andrew showed me how I could speed up and slow down, and the computer stayed with me. I was impressed although it sometimes felt as if I had to coax it into changing speed a little with very clear notes, but that's true of many bands I've played with too!

What is your opinion of the musical parts that the accompanist plays and the sound quality of the instruments?

It's all played very simply, which is fine but it could do with a little variation. The instruments do sound very artificial, but it's good enough to hear what's going on.

To what extent do you think that the accompanist system plays like a human?

I find it a little hard to judge because the instruments don't sound enough like real instruments to fool me. In terms of what it plays though it certainly listens to what I play in a pretty human way, the way it speeds up and slows down is the most human element.

Is there any functionality that you would like to add?

It would be nice if the instruments sounded a little more realistic.

If a commercial product along these lines was released would you be interested in using or buying it?

I'd certainly be interested in giving it a try. A lot of pubs have licences limiting them to only two live musicians, and I know this forces a lot of people to play with bass and drums on a backing track, so there's definitely potential for a system like this there. Makes playing by yourself a bit more fun too.

Any further comments?

I'm impressed that a computer can do this, and on the basis of what Andrew's done in a few months I think there's a lot of potential for the future, I'm sure we'll see more of this kind of thing!

5.4.2 User 2 – Distorted Electric Guitar

This user test can be heard on the audio c.d. track 3

Could you tell us which instrument you play, and give us an idea of your experience of playing blues music?

I've played guitar for about 5 years and a blues musician taught me, so I love playing blues.

How successful do you think the accompanist was at coming in on time and in tune?

It came in well, sounds perfectly in tune so no complaints.

Did the accompanist stop at the right time and did its ending sound good?

Yes the ending sounded fine.

How successful do you think the accompanist was at staying in time and tune throughout the song?

It seemed to stay with me. I had a go at speeding up and slowing down and it stayed with me pretty well.

What is your opinion of the musical parts that the accompanist plays and the sound quality of the instruments?

They sound more like midi than real instruments but fine to play along to.

To what extent do you think that the accompanist system plays like a human?

It seems to hear what I play and change what it plays quite realistically. I don't think that I'd actually believe it was a human playing, but it shows signs of being on the right lines.

Is there any functionality that you would like to add?

It would be nice for it to be able to play different chord structures for different types of music too.

If a commercial product along these lines was released would you be interested in using or buying it?

Depending on the cost and what it sounded like I think I would.

Any further comments?

Having never seen anything like this before I think it's a great initial effort and it would be nice to see it developed further.

6 Conclusion

The overall aim of this project was to develop a real-time blues accompaniment system that modelled human accompaniment. Within the constraints of the development time this aim has been met with a fair degree of success. The accompanist system successfully plays along with a lead musician in time and tune, and shows at least some of the characteristics of human musicianship.

The initial requirements will be assessed, and to what degree they have been met evaluated in the 'Meeting Requirements' section below. Any areas of the requirements that were not fully met will be discussed further as part of the 'Critical Appraisal' section. Further ideas of how the system could be developed in the future will then be discussed in the 'Further Work' section.

6.1 Meeting Requirements

Start the song correctly along with the lead musician. The count-in analysis method was successful to a point at which the system came in on time, at the right speed, and at the right pitch most of the time. This requirement has therefore been met to a reasonable degree.

Play at least the minimal accompanying instruments. The system not only plays basic percussion and bass notes, but also a drum kit, a bass line, and a rhythm guitar, so this basic requirement was met fully.

Change key according to the twelve bar blues chord sequence at the correct times. By using frequency ratios the system was able to calculate the blues scale and chord changes for any initial pitch, and it changed chords correctly for the key that the song was in, so again this requirement was met.

End at an appropriate time. In this system's final state it waited until there was enough silence from the musician to assume that they had finished, and then stopped after the next twelve bar part. This was adequate in that the system did stop, but the method of doing so could have been improved upon.

Stay in time throughout the song. After much trial and improvement in the beat tracking and correction competencies, the system made a good, and reasonably human like effort to react to tempo changes in the lead musician's playing. Although the requirement was generally met there were some flaws. The RMS based onset detection method used meant that notes with little amplitude change at their start were not recognised all of the time. Also if there was a loud note ringing as the beat was approached, it was difficult for the system to distinguish it from a new note, and so again the note would not be identified in every case.

Stay in tune throughout the song. The method implemented for recognising the key of the song from the musician's count-in was fairly successful. Any errors in this initial pitch assumption were soon corrected by the root note recognition during the song. Chord changes were correct and at the right times. The system did not, however, allow for the lead musician changing the key of the entire song during the performance. This requirement was therefore met adequately but not perfectly.

Provide a realistic sounding full blues backing band. This requirement is at least partially down to personal opinion and taste. The output from the system did sound like a bass guitar, drum kit and guitar playing blues backing, but the instruments sounded computer generated and a little repetitive. While the basic requirements have been surpassed in this area, the band still did not sound ‘realistic’, and most people would have no trouble in distinguishing it from genuine instruments.

Be able to recognise when the musician is playing a more unusual blues styles such as eight-bar, and change chords appropriately. Due to time constraints this requirement was not addressed in practice at all. An accompaniment system that could do this could in fact be adapted to suit many other styles of music, an area that is discussed in further detail in the ‘Further Work’ section.

6.2 Critical Appraisal

An appraisal of the processes used to accomplish the final system was given in the ‘Evaluation’ section of this document. Here we will look critically at the final product that was reached. The reactions of the users that tested the system, and the feedback received from them, will be taken into account here.

There were obviously some weaker areas in the system that it would have been preferable to have spent more time on. Many of these areas were highlighted by the user testing.

The onset detection method used was a little over simplistic; the detection threshold had to be re-calibrated for different instruments or volume levels, and it did occasionally miss beats even when configured correctly. This is because it relied entirely on a distinct amplitude peak at the onset of a note, something that was not always present. In reality an onset detection method that looked for changes in either amplitude or pitch might have been more consistent in picking up notes, and therefore allowed the system to react more consistently.

Although it is unusual for the entire key of the song to be changed mi-performance in blues music, it is something that would have been interesting to allow for. The ability to do this would lead easily to further ideas of versatility with the type of music that the accompanist could work with, and would be a step towards fully modelling a human musician. It would also be nice for the lead musician to be able to correct themselves if they accidentally played the wrong note in the count-in, and this ability would allow for this.

The instrument modelling in this system was very basic, and had some negative feedback from the user tests. While adequate for a prototype system the instruments did not genuinely sound like the real thing. The user’s initial impressions would have been improved greatly if the project had been stronger in this field.

The musical parts played by each of the backing instruments are not particularly varied, and by improving in this area, the system would sound more human still. To make the system truly varied in what it played, some elements of computer composition would have to be introduced into the system. Some ideas surrounding this are discussed in the 'Further Work' section. The ending of the songs is again a little weak, and again while adequate for a prototype, could be improved upon.

The user testing of the system was critical, and gave the best overall indication of the success of the system. The project would in fact have benefited greatly from more real world testing, particularly with more different types of instruments, and on more different computer systems.

Despite the criticisms discussed, there has been a lot of success in the development of this system, and its overall aims have been met. Potential for what could be achieved in this field has been demonstrated, and a working intelligent accompanist system has been produced. Although the system is still extremely basic when compared to its human counterparts, it has begun to show at least some of their behaviour in the way it reacts. It is a project that is interesting both from the theoretical viewpoint of modelling human behaviour, and the practical viewpoint of making playing alone more interesting.

6.3 Further Work

6.3.1 Advanced Instrument Modelling

During this project the intelligence of the accompanist was the main focus, and only very basic models of the instruments it would play were used. As test users commented, they did sound a little artificial. The physical modelling and sample based synthesis fields are currently very advanced, and some truly realistic sounds can be generated. Some of these methods were covered in the literature survey. Using more advanced techniques such as these would definitely improve the sound and overall realism of the system. Having realistic sounding instruments would make the whole system appear more human, and therefore improve the initial impression users got from the system.

6.3.2 Algorithmic Composition

In order to really fully replicate a human blues musician, even if they are only playing the role of accompanist, some degree of improvisation would have to be modelled.

An advanced blues bass player for instance, will rarely stick to the same bass line all the way through a song, and will certainly not play the same bass line in every song. This system currently plays a bass line on the correct scale, changing scales correctly with chord changes, but it does not choose the notes on that scale with any intelligence or even variation. As one of the test users commented, this does make it sound a little repetitive.

Elements can be taken from the field of algorithmic composition to start to model the way human players will improvise musical parts, and in creating a system that could actually be considered as good as a human this would be an essential extra element to add.

6.3.3 Different Musical Styles

Blues music was chosen for this system, not only because it is the author's personal area of interest, but also because it is intrinsically an easy musical style to accompany. Chord progressions are predictable; once you are playing in the right key and you know where you are in the song, the correct chord is fairly easily derived.

Many of the principles used in this system would, however, be perfectly applicable to other musical styles. Essentially all that needs to be changed is the musical lines that the generated instruments follow, and the way chord changes are calculated.

Now in some musical styles working out these chord changes in advance with no prior knowledge of the song could be very difficult. That is not to say that a useful accompanist could not be constructed however. What one needs to do is to think about the way that a musician might try and learn a pop song from the radio. If the song is at all repetitive in nature then the musician will make fewer and fewer mistakes as the song goes on, as they will learn chord changes in the song as they go. When the musician does make a mistake they will quickly correct it.

A system to model this would need to try to predict the correct chord change, and correct itself as quickly as possible when it was wrong. It could then learn from these mistakes, and presented with the same conditions in the same song again it would have a better idea of the right chord to change to. If this knowledge could then be stored, and the system heard the same song again then it would already know all the chord changes. In this way a lead musician could 'teach' the accompanist their repertoire of songs until it could play them all correctly. The system could also in this way become more adept in certain genres of music where certain chord structures occur more often.

This would be a fascinating step forward in computer music accompaniment and would also be very interesting from the point of view of artificial intelligence and agent design.

6.3.4 Improved Ending Recognition

After an analysis of blues musicians approaching song endings, the following observations were noted:

- Melodies tend to converge towards the root note, more often descending but sometimes ascending.

- Notes are often held for longer so that the melody becomes slower even if the tempo is the same.

- Tempo will not necessarily change at the end of a song. If tempo does change it is usually slows down, but does occasionally speed up.

- If the tempo does change it will do so far more rapidly than normal, so the tempo adjustment algorithm of the accompanist would have to react differently towards an ending.

- Volume is very likely to change. Songs can either finish on a loud crescendo, or fade to a quiet closing. Slower songs seem more likely to finish quietly.

Now there are exceptions to each and every one of the observations above, and it is often subtle combinations of each that allow us to detect an approaching ending. Attempting to build this competency into the system would certainly be a challenge, but it would be a very interesting one and would make the system feel all the more human.

6.4 Personal Reflection

I have found this project really challenging, but I am genuinely proud of the finished product. I think that a system like this, once a little more polished, could be a real commercial success. It is something that has not been done in this way before, and the reactions I have had from people to whom I have shown the system have been really good.

I was especially pleased to read an email from Dr. Richard Boulanger (author of 'The CSound Book' and expert in the field) to Prof. John Fitch in which he stated: "This is really great! What a super design and a really cool practice system…. Congratulations to you and your student." To have such a positive reaction from someone so prominent in this field makes the hard work worthwhile.

The system itself is developed to a point where it is fun to use, which was always a personal aim of mine. I find that I have continued to use it ever since completing it, it really does make playing my guitar alone more interesting. I have even experimented with recording some songs using it, one of which is on the included audio c.d. Having a real interest in the project really drove me to put extra time and effort into it, and it is something that I will be proud of for many years to come. I hope that is conveyed in this dissertation.

7 Bibliography

Boulanger, R (2000). ‘The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming’. The MIT Press.

Brooks, RA (1989). ‘A Robot that Walks; Emergent Behaviours from a Carefully Evolved Network’. MIT Artificial Intelligence Laboratory.

Brossier PM, (2005). ‘Fast Onset Detection Using Aubio’, MIREX Contest Submission.

Bryson, J, (1992). ‘The Subsumption Strategy Development of a Music Modelling System’. Department of Artificial Intelligence, University of Edinburgh.

Dannenberg, R. (1984). ‘An On-Line Algorithm for Real-Time Accompaniment’. Proceedings of the International Computer Music Conference, Paris, 193 - 198.

Davis, F. (2003). ‘The History of the Blues: The Roots, the Music, the People’. Da Capo Press.

De la Cuadra, P (2001). ‘Efficient Pitch Detection Techniques for Interactive Music’, Proceedings of International Computer Music Conference.

Gerhard, D (2003). ‘Pitch Extraction and Fundamental Frequency: History and Current Techniques’. Department of Computer Science, University of Regina.

Goto, and Muraoka, (1998). ‘Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions’. School of Science and Engineering, Waseda University.

Horner, A., Beauchamp, J., and Haken, L. (1993). "Methods for Multiple Wavetable Synthesis of Musical Instrument Tones," Journal of the Audio Engineering Society, 41(5), 336-356.

Karplus, K, and Strong, A, (1983). "Digital Synthesis of Plucked String and Drum Timbres". Computer Music Journal 7 (2): 43-55.

Piszczalski, M (1986). ‘A Computational Model for Music Transcription’ PhD thesis, University of Stanford.

Scheirer ED, (1997). ‘Pulse Tracking with a Pitch Tracker’, Proceedings of Workshop on Applications of Signal Processing to Audio and Acoustics.

Wooldridge, M (2002). ‘Introduction to MultiAgent Systems’. John Wiley & Sons.

Ying, Jamieson and Michell (1996). ‘A Probabilistic Approach to AMDF Pitch Detection’, Proceedings Fourth of International Conference on Spoke Language.

8 Appendices

8.1 Audio CD Contents

Included with the project are some recordings of the system working. These are both recorded to an audio c.d. and included with the other project files on the data c.d. Below is a description of each track.

1 – Basic Use. A simple example played on guitar demonstrating how the system is counted in, how it stays in time when the musician changes tempo, and how it ends.

2 – User 1 Test. A recording of the test carried out by the first user.

3 – User 2 Test. A recording of the test carried out by the second user.

4 – Complete Song (with overdubs). A full blues song with vocals and harmonica overdubbed.

5 – Complete Song (original recording). A recording of the original performance that the song was built from.

8.2 Program Code

8.2.1 accompanist.orc

This is the ‘orchestra’ CSound code which contains all the main functionality of the program. All the functional code is split into separate ‘instruments’. The first of these deals with the analysis of the input. The second deals with the generation of the backing music output. All following instruments are actually the instrument models.

```
; global configuration variables
sr = 10000
kr = 125
ksmps = 80
nchnls = 1

; global user configurable variables
gkonsetlev init 1500  ;set depending on input level
gkoutlatency init 50  ;allows for latency in pause after count in
gkinlatency init 13   ;allows for latency in expected beat
position for in song tempo

; global variables counting for tempo recognition
gkfirstcount init 0
gklastcount init 0
gkcountnumber init 0
gktempo init 0
gktempochange init 0
gkchecktempo init 0

; global variable for pitch
gkpitch init 0

;global variables for song position
gkbar init 1
gkbeat init 1

;global variable waiting for song end
gksilence init 0


;-------INSTRUMENT 1 recognises tempo and pitch of input----------
instr 1
  ;current time
  ktime timek

  ;mic input
  asig in
  ;apply a lowpass-filter
  asig tone asig, 1000

  ;the rms of current input
  krms rms asig

  ;-------------DURING THE COUNT IN----------------

  if (gkcountnumber <= 3) then
```

```
    ;if it's the first note played start counting
    if (krms > gkonsetlev && gkcountnumber == 0) then
      kbeatpitch, krms pitchamdf asig, 40, 80;, 55, 0, 3
      gkpitch = gkpitch + kbeatpitch
      gkfirstcount = ktime
      gklastcount = ktime
      gkcountnumber = gkcountnumber + 1
      printk 0,gkfirstcount

    ;if it's the second or third keep counting
    elseif (krms > gkonsetlev && gkcountnumber < 3 && ktime >
gklastcount+(kr/2)) then
      kbeatpitch, krms pitchamdf asig, 40, 80;, 55, 0, 3
      gkpitch = gkpitch + kbeatpitch
      gklastcount = ktime
      gkcountnumber = gkcountnumber + 1
      printk 0,gklastcount

    ;if the count isn't completed within 10 seconds start over
    elseif (gkcountnumber > 0 && ktime > gklastcount+kr*2) then
      gkcountnumber = 0
      printks "Restarting the count in",0

    ;if it's the fourth and final count then extract tempo and
pitch and generate chords
    elseif (krms > gkonsetlev && gkcountnumber == 3 && ktime >
gklastcount+(kr/2)) then

      ; extract pitch and envelope of current input
      kbeatpitch, krms pitchamdf asig, 40, 80;, 55, 0, 3
      gkpitch = (gkpitch + kbeatpitch)/4
      gklastcount = ktime
      gkcountnumber = gkcountnumber + 1

      ;calculate the average time between the intro beats
      gktempo = int((gklastcount - gkfirstcount)/3)
      printk 0,gktempo

    endif

  endif

  ;----------------AFTER THE COUNT IN-------------------

  if (gkcountnumber > 3) then

    ;if the count in is complete and it's exactly the start of a
beat trigger backing music
    if (ktime == (gklastcount + gktempo - gkoutlatency)) then
      gklastcount = ktime + gkoutlatency
      gktempo = gktempo + gktempochange
      gktempochange = int(gktempochange/2)

      ;increment the silence count
      gksilence = gksilence + 1

      ;if a root note is being played fine tune the pitch
      kpitchcheck, krms pitchamdf asig, 40, 80, 55, 0, 3
      if (kpitchcheck < gkpitch*1.03 && kpitchcheck > gkpitch*.97)
then
       ;take the average of the believed and the new
       gkpitch = (kpitchcheck + gkpitch) / 2
```

```
     endif

     ;call the instrument that generates the backing music
     event "i", 2, 0, 1/kr

   ;if count is complete and it's close to the start of a beat
check we're in pitch and in time
   elseif (ktime>=(gklastcount+gkinlatency-10) &&
ktime<=(gklastcount+gkinlatency+10)) then

     ;if we're at the start of the window and there's not already
a note ringing
     if (ktime==(gklastcount+gkinlatency-10) && krms>gkonsetlev)
then
      ;stop any more checking this beat
      gkchecktempo = 0

     ;once inside window if a note is played adjust tempo
accordingly
     elseif (gkchecktempo==1 && krms>gkonsetlev) then
      printk 0,ktime-(gklastcount+gkinlatency)
      ;adjust tempo
      gktempochange = gktempochange + ktime-
(gklastcount+gkinlatency)
      ;stop any more checking this beat
      gkchecktempo = 0

     ;once at the end of the window
     elseif (ktime==(gklastcount+gkinlatency+10) &&
gkchecktempo==0) then
      ;end of check, reset gkchecktempo so it starts the check
again next beat
      gkchecktempo = 1
      ;there has been sound so reset the silence count
      gksilence = 0
     endif

   endif
  endif

endin

;----INSTRUMENT 2 generates correct backing music at each beat----
instr 2

    ;convert tempo to beat length in seconds
    kbeatlength = gktempo/(2*kr)

    ;change chord according to current bar if appropriate
    if (gkbar < 5 || gkbar == 7 || gkbar == 8 || gkbar ==11 ||
gkbar == 12 ) then
     kchord = gkpitch
    elseif (gkbar == 5 || gkbar == 6 || gkbar == 10 ) then
     kchord = gkpitch * 1.333
    elseif (gkbar == 9) then
     kchord = gkpitch *1.5
    endif

    ;play correct instruments for current beat
    if (gkbeat == 1) then
     ;bass guitar
     event "i", 3, 0, kbeatlength, 5000, kchord
     event "i", 3, kbeatlength, kbeatlength, 5000, kchord *1.26
```

```
 ;guitar
 event "i", 7, 0, kbeatlength, 10000, kchord*4
 event "i", 7, 0, kbeatlength, 10000, kchord*6
 event "i", 7, 0, kbeatlength, 10000, kchord*8
 event "i", 7, 0, kbeatlength, 10000, kchord*12
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*4
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*6.73
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*8
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*13.46

 ;drums
 event "i", 4, 0, .4, 500
 event "i", 5, 0, .1, 300
 event "i", 5, kbeatlength, .1, 300

elseif (gkbeat == 2) then
 ;bass guitar
 event "i", 3, 0, kbeatlength, 5000, kchord*1.5
 event "i", 3, kbeatlength, kbeatlength, 5000, kchord *1.68

 ;guitar
 event "i", 7, 0, kbeatlength, 10000, kchord*4
 event "i", 7, 0, kbeatlength, 10000, kchord*6
 event "i", 7, 0, kbeatlength, 10000, kchord*8
 event "i", 7, 0, kbeatlength, 10000, kchord*12
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*4
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*6.73
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*8
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*13.46

 ;drums
 event "i", 5, 0, .4, 800
 event "i", 5, 0, .1, 300
 event "i", 5, kbeatlength, .1, 300

elseif (gkbeat == 3) then
 ;bass guitar
 event "i", 3, 0, kbeatlength, 5000, kchord*1.78
 event "i", 3, kbeatlength, kbeatlength, 5000, kchord *1.68

 ;guitar
 event "i", 7, 0, kbeatlength, 10000, kchord*4
 event "i", 7, 0, kbeatlength, 10000, kchord*6
 event "i", 7, 0, kbeatlength, 10000, kchord*8
 event "i", 7, 0, kbeatlength, 10000, kchord*12
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*4
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*6.73
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*8
 event "i", 7, kbeatlength, kbeatlength, 10000, kchord*13.46

 ;drums
 event "i", 4, 0, .4, 500
 event "i", 4, kbeatlength, .4, 500
 event "i", 5, 0, .1, 300
 event "i", 5, kbeatlength, .1, 300

elseif (gkbeat == 4) then
 ;bass guitar
 event "i", 3, 0, kbeatlength, 5000, kchord*1.5
 event "i", 3, kbeatlength, kbeatlength, 5000, kchord *1.26

 ;guitar
```

```
     event "i", 7, 0, kbeatlength, 10000, kchord*4
     event "i", 7, 0, kbeatlength, 10000, kchord*7.13
     event "i", 7, 0, kbeatlength, 10000, kchord*8
     event "i", 7, 0, kbeatlength, 10000, kchord*14.26
     event "i", 7, kbeatlength, kbeatlength, 10000, kchord*4
     event "i", 7, kbeatlength, kbeatlength, 10000, kchord*6.73
     event "i", 7, kbeatlength, kbeatlength, 10000, kchord*8
     event "i", 7, kbeatlength, kbeatlength, 10000, kchord*13.46

     ;drums
     event "i", 5, 0, .4, 800
     event "i", 5, 0, .1, 300
     event "i", 5, kbeatlength, .1, 300

    endif


    ;move to next beat/bar
    gkbeat = gkbeat + 1

    ;if it's the last beat of the bar
    if (gkbeat == 5) then
      ;reset beat to 1
      gkbeat = 1
      ;move to next bar
      gkbar = gkbar + 1

      ;if it's the last bar
      if (gkbar == 13) then
        ;let root note ring and finish if there has been no sound
for a period
        if (gksilence >= 4) then
         event "i", 3, kbeatlength*2, kbeatlength*8, 5000, kchord
         event "i", 7, kbeatlength*2, kbeatlength*8, 10000,
kchord*4
         event "i", 7, kbeatlength*2, kbeatlength*8, 10000,
kchord*6
         event "i", 4, kbeatlength*2, .4, 500

         printk 0,gksilence
         ;setting tempo to 0 to terminate further backing music
         gktempo=0
        endif
         ;reset bar to 1
         gkbar = 1

      endif
    endif

endin


;-------INSTRUMENT 3 BASS-----------------------------------------
instr 3

  ; a very simple synthesised waveform
  asig1 oscil p4, p5, 1

  out asig1
endin

;-------INSTRUMENT 4 KICK DRUM------------------------------------
instr 4
```

```
  iamp      = p4

  k1  expon     120, .2, 50
  k2  expon     500, .4, 200
  a1  oscil     iamp, k1, 1
  a2  reson     a1, k2, 50
  a3  butterlp  a2+a1,k1,1
  a4  butterlp  a3,   k1,1
  a5  butterlp  a4,2500,1
  a6  butterhp  a5,50
  a7  butterhp  a6,50
  a8  linen     a7,0.01,p3, .2

  out a8
endin

;-------INSTRUMENT 5 SNARE DRUM-----------------------------------
instr 5
  iamp     = p4

  kamp1 expon 1.25, .03,  .0001
  kamp2 expseg .001,.005,1, .35, .001

  anoise rand 1

  anoise1 = anoise*kamp1
  anoise2 = anoise*kamp2

  adel1     = anoise1
  adel2 delay anoise1, .01
  adel3 delay anoise1, .02

  adel4 delay anoise2, .03

  abp1 resonz adel1,  400,  1100
  abp2 resonz adel2,  600,  1100
  abp3 resonz adel3,  800,  1100
  abp4 resonz adel4,  1100, 1100

  out iamp*2*(abp1+abp2+abp3+abp4)
endin

;-------INSTRUMENT 6 HI HAT---------------------------------------
instr 6

  ilen = p3
  iamp = p4/100000

  kcutfreq  expon     10000, 0.1, 2500
  aamp      expon     iamp,  0.1,   10
  arand     rand      aamp
  alp1      butterlp  arand,kcutfreq
  alp2      butterlp  alp1,kcutfreq
  ahp1      butterhp  alp2,3500
  asigpre   butterhp  ahp1,3500
  asig      linen    (asigpre+arand/2),0,ilen, .05

  out asig

endin

;-------INSTRUMENT 7 GUITAR---------------------------------------
```

```
instr 7

  kamp linseg 0.0, 0.015, p4*2, p3-0.065, p4*2, 0.05, 0.0
  asig pluck  kamp, p5, p5, 0, 1
  af1  reson  asig, 110, 80
  af2  reson  asig, 220, 100
  af3  reson  asig, 440, 80
  aout balance 0.6*af1+af2+0.6*af3+0.4*asig, asig

  out aout

endin
```

8.2.2 accompanist.sco

Most CSound programs will use the ‘score’ section of the code to specify the structure of the song; when and how to play each instrument. As the music is generated dynamically in real time from the ‘orchestra’ section of the code, this is not necessary in this system. All that is in this file is a wave table used for some of the instrument modelling, and a command to run the main controlling instrument (i1) in the orchestra file for up to ten minutes.

```
;A sine wave table used for various instruments
;Table  Start TableSize TableGenerator Parameter
f1      0     65536     10             1

;just play controling instrument for a set time
;Instrument(p1)  Start(p2) Duration(p3)
i1               0         600
```